Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática

Escuela Académico Profesional de Ingeniería de Sistemas

Integración de aplicaciones empresariales mediante el enterprise service bus, caso de uso: Programa Nacional de Apoyo Directo a los Más Pobres-PCM

TESINA

Para optar el Título Profesional de Ingeniero de Sistemas

AUTOR

Oscar ALVARO AGUILAR

ASESOR

Armando FERMÍN PÉREZ

Lima, Perú

2010

Referencia bibliográfica

Alvaro, O. (2010). *Integración de aplicaciones empresariales mediante el enterprise service bus, caso de uso: Programa Nacional de Apoyo Directo a los Más Pobres-PCM.* Tesina para optar el título profesional de Ingeniero de Sistemas. Escuela Académico Profesional de Ingeniería de Sistemas, Facultad de Ingeniería de Sistemas e Informática, Universidad Nacional Mayor de San Marcos, Lima, Perú.

"Tesis presentada a la Universidad Nacional Mayor de San Marcos, Lima, Perú, para obtener el Título de Ingeniero de Sistemas"

Orientador: Armando Fermin Perez

DEDICATORIA

Este trabajo esta dedicado a toda mi familia.

RESUMEN

En la actualidad el área de tecnología de información se enfrenta a entornos cambiantes orientados por los cambios tecnológicos y organizaciones dinámicas, requiriéndose respuestas en tiempos aceptables, dentro de costos estimados y con productos de calidad, este contexto a motivado a la creación de un conjunto de sistemas en distintas arquitecturas, lenguajes, base de datos y software base, obteniendo distintas aplicaciones para dar soporte a los procesos de negocio que son cambiantes en el tiempo, tornando compleja la administración de los sistemas.

La solución propuesta a la problemática es el uso de una capa común a todas las aplicaciones la cual se denomina Bus de Servicios Empresariales (ESB por sus siglas en ingles), quien se encargará de estandarizar la comunicación que viaja en él y orquestar las funcionalidades del software para dar soporte a los procesos del negocio.

En el capitulo uno, se describe la problemática que presenta el área de informática frente al manejo de diferentes aplicaciones en un entorno de negocio cambiante; en el capitulo dos, se presenta el marco teórico acerca del Bus de Servicios Empresariales y los distintos componentes que lo integran; en el capitulo tres, se describe el Java Business Integration (JBI), como especificación para el desarrollo del ESB sobre la cual se integran las aplicaciones empresariales; en el capitulo cuatro, se realiza la selección de herramientas software y la forma que se aplican para el desarrollo del ESB; en el capitulo cinco, se diseña el motor de servicio y el componente de vinculación del ESB; en el capitulo seis, se muestran los resultados obtenidos al aplicar el ESB en el Programa Nacional de Apoyo Directo a los Más Pobres - PCM.

Palabras Claves: integración de aplicaciones, ESB, arquitectura abierta.

ABSTRACT

At present the area of information technology is facing changing environments guided by technological change and dynamic organizations, requiring answers in acceptable time, within estimated costs and quality products, this context led to the creation of a set of systems on different architectures, languages, database and software base, getting different applications to support business processes that are changing over time, making complex systems management.

The proposed solution to the problem is the use of a common layer to all applications which is called Enterprise Service Bus (ESB), who will standardize communication that travels on it and orchestrate functionalities software to support business processes.

In Chapter one describes the issues presented by the area of information against the handling of different applications in a changing business environment, in chapter two, we present the theoretical framework about Enterprise Service Bus and the various components that up, in the third chapter describes the Java Business Integration (JBI) as a specification for the development of ESB on which business applications are integrated, in chapter four, they choose the software tools and the manner applied to the development of the ESB; in chapter five, the engine is designed and the service linking the ESB component, in chapter six, shows the results obtained by applying the ESB in the National Program of Direct Support for More Poor - PCM.

Key words: composition of applications, ESB, open architecture.

ÍNDICE

Lista de figuras

Lista de tablas

Capítulo 1: Introducción

1.1 Antecedentes

"JUNTOS es un Programa Social dirigido a la población de mayor vulnerabilidad, en situación de extrema pobreza, riesgo y exclusión. El programa tiene como objetivo promover el ejercicio de sus derechos fundamentales a través de la articulación de la oferta de servicios en nutrición, salud, educación e identidad. Para lograr este objetivo JUNTOS entrega un incentivo monetario condicionado de libre uso para la/el representante (madre, padre) de cada hogar participante." [1].

Figura 1. Condicionalidades a cumplir

En la actualidad los negocios son cambiantes, evolucionan rápidamente y no son aislados, interactúan con otras organizaciones, en este contexto las tecnologías de información deben de responder con la rapidez y con productos de calidad aprovechando al máximo la colaboración y sin desechar funcionalidades existentes en sistemas implantados durante años.

[1] Decreto Supremo 055-2007-PCM.

En este contexto JUNTOS no está aislada de la definición anterior, debido que para el logro de sus objetivos estratégicos, debe de interactuar con otros sectores para poder evaluar el cumplimiento de condiciones de los hogares beneficiarios, lo que conlleva al problema de compartir información con otras instituciones y tener distintas aplicaciones para soportar los macroprocesos que tiene la organización, muchas de ellas construidas en casa y otras tercerizadas.

1.2 Definición del problema

El principal problema que presenta la organización (JUNTOS), para el cumplimiento de sus funciones depende de un marco legal que es cambiante y dependiente de la coyuntura nacional, para colaborar con otras instituciones se realiza mediante convenios interinstitucionales cuya esencia es compartir información de los beneficiarios de forma oportuna ; para dichos cambios se iban realizando soluciones software que no se integran por completo en un macro sistema para dar soporte a los procesos core del negocio.

Las aplicaciones desarrolladas se encuentran en distintas plataformas, arquitecturas de software, lenguajes de programación, sistemas operativos.

El intercambio de información con otras instituciones se realiza mediante Base de Datos y Web Service que se aloja en los servidores de cada institución.

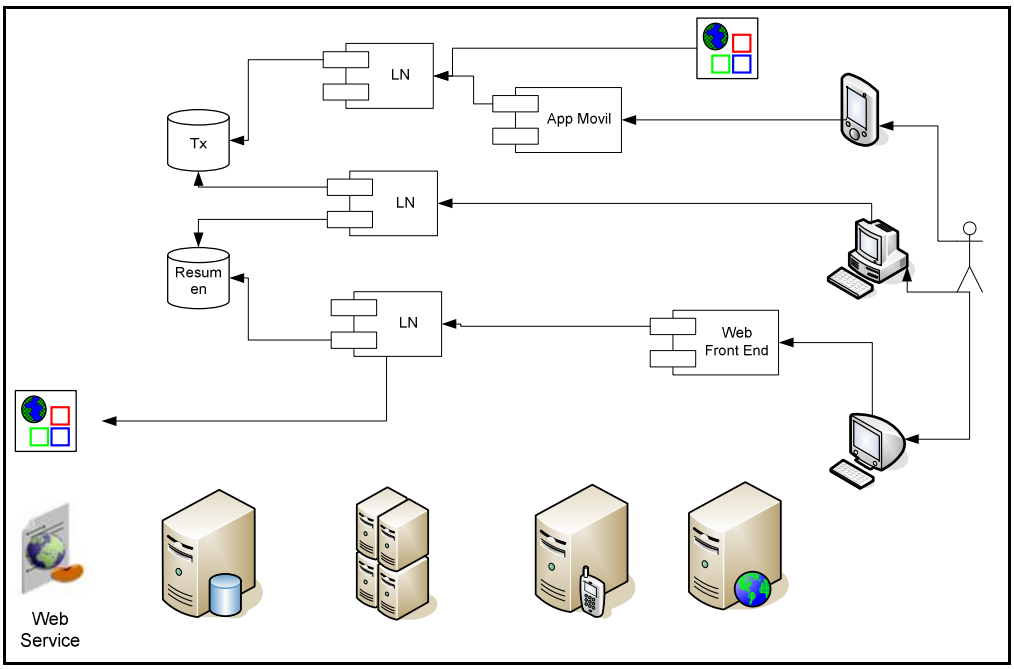


Figura 2. Arquitectura actual de la Institución

En la figura 2, se aprecia que la lógica del negocio está dispersa en distintas aplicaciones y en algunos componentes la lógica presenta redundancia de funcionalidades, tornando complejo el mantenimiento, por ejemplo: si se desea modificar la funcionalidad de un caso de uso que se encuentra en dos aplicaciones se debe modificar las dos aplicaciones, en caso se omita la modificación de uno de los componentes del sistema puede presentar inconsistencia en los datos y un comportamiento no uniforme de todo el sistema.

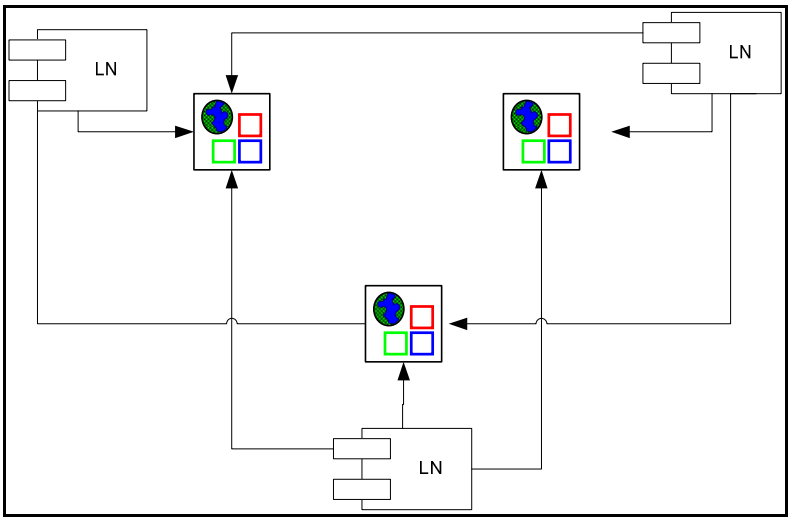


Figura 3. Consumo y producción de servicios web

En la figura 3, se indica como es la producción y consumo de los servicios web, para que los servicios sean interoperables es necesario que cada componente del sistema se comunique con todos los servicios necesarios para cumplir su funcionalidad, esto repercute en un despliegue complejo.

1.3 Justificación

Dentro del ciclo de vida del software la actividad que consume más tiempo es el mantenimiento debido al acoplamiento de los componentes del sistema, además que la lógica está dispersa y redundante entre una aplicación y otra, debiendo revisar la documentación y diagramas de las distintas aplicaciones para realizar mantenimientos en el macro sistema que soporta los procesos del negocio. Para que este mantenimiento se reduzca en costo-tiempo es necesario tener los procesos de negocio

centralizados de esta forma los datos serán menos propensos a errores de inconsistencias. No se desecha por completo sistemas legados, ya que se puede conectar la funcionalidad útil con el ESB y de esta forma se reducen costos y esfuerzo.

El problema de la integración de aplicaciones se está convirtiendo en una necesidad común entre las medianas y grandes organizaciones donde la cooperación interinstitucional es vital para su funcionamiento y la gestión del negocio se inclina hacia la gestión por procesos ante la ineficiencia de las organizaciones departamentales.

1.4 Objetivos

1.4.1 Objetivo principal

Diseñar el Enterprise Service Bus que permita integrar las distintas aplicaciones de la organización (JUNTOS) y los servicios proporcionados por otras instituciones (salud, educación e identidad), utilizando para este fin la especificación Java Business Integration (JBI).

1.4.2 Objetivos secundarios

- Revisar la arquitectura de los sistemas actuales y definir cuales se pueden unir al ESB.
- Diseñar los medios de comunicación estándar (protocolos) entre las diversas aplicaciones y servicios de terceros.
- Transformar los mensajes que viajen en el ESB para el común entendimiento de las aplicaciones.
- Crear un capa Proxy por encima de los servicios, ello permitirá abstraer la implementación y agregarle información o requisitos para su ejecución.
- Definir políticas de seguridad para los servicios a implementar y establecer requisitos para la invocación y ejecución de estos.

1.5 Propuesta de la tesis

La integración de aplicaciones se realizará mediante el ESB utilizando la especificación Java Business Integration, esta integración se refiere a protocolos, plataformas y forma de accesos como http, ftp, sockets, etc. Para el logro de este fin se iniciará revisando las aplicaciones y arquitecturas software existentes en la organización (JUNTOS), luego se crean las interfaces mediante web service (principalmente) que se conectarán (plug in) al ESB mediante el Binding Components que es el responsable de la comunicación con protocolos remotos donde se normalizan y desnormalizan mensajes usando un Binding Component especifico para cada protocolo: http, ftp, RMI, JMS, etc. Estos servicios creados por las distintas aplicaciones estarán orquestados por el Service Engine que provee la secuencia de servicios a consumir para atender un proceso de negocio, mediante un motor de reglas de negocio, motores BPEL, motores XSLT, motores de script y EJB containers, en el presente trabajo se utilizará principalmente los motores BPEL que presenta una notación parecida al BPMN muy utilizada para modelar procesos de negocios. Para el enrutamientos de los mensajes entre aplicaciones se hace uso Normalized Message Router que se encuentra dentro del mismo ESB que utiliza la especificación JBI que proporciona transparencia de ubicación a los servicios.

Cabe resaltar que utilizar un ESB no implica que se esté implementando SOA, esté es un concepto más amplio donde el ESB es el componente principal pero no el único. SOA depende de la forma que se modela y conciben los sistemas que luego orquestarán para suplir las necedades del negocio.

1.6 Presentación del resto de la tesis

En el Capítulo 2 se aborda el marco teórico, se presentan los diversos conceptos utilizados en el ESB como Binding Components, Service Engines, Normalized Message Router, también se muestra la arquitectura de las aplicaciones existentes en la institución (JUNTOS).

En el Capítulo 3 se aborda el estado del arte, acá se detallan las distintas herramientas que permiten la integración de aplicaciones software y como fueron aplicados en distintos proyectos.

En el Capítulo 4 se realiza el aporte teórico, adaptando los conceptos en la solución integradora para JUNTOS, formulado en el capítulo de introducción.

En el Capítulo 5 se desarrollan los artefactos necesarios para el diseño del ESB.

En el Capítulo 6 se muestra los resultados al aplicar el ESB dentro de JUNTOS, estos resultados son plasmados mediante indicadores que muestran el desempeño del desarrollo de software antes y después de la aplicación del ESB.

En el Capítulo 7 se muestran las conclusiones al desarrollar la integración de aplicaciones mediante el uso del ESB con la especificación JBI y como esto nos abre las puertas para un futuro uso de SOA y de la perfecta alineación con los negocios gestionados por procesos.

Capítulo 2: Marco Teórico

Se listarán los distintos conceptos genéricos que se utilizan en el diseño de un Enterprise Service Bus, que apoyaran al desarrollo de la solución al problema formulado en la introducción, sin estar enmarcado en el uso de una herramienta especifica.

2.1 Software distribuido

El desarrollo orientado a servicios (SOA) comparte varios de los principios de las aplicaciones distribuidas orientadas a objetos, como puede ser el encapsulamiento, la abstracción, las interfaces claramente definidas.

2.1.1 Cliente Servidor

Es un sistema donde el cliente tiene toda la lógica de negocio y acceso a datos y el servidor suele ser la únicamente un repositorio de información.

El servidor puede ser un repositorio de archivos, un servidor de base de datos, etc.

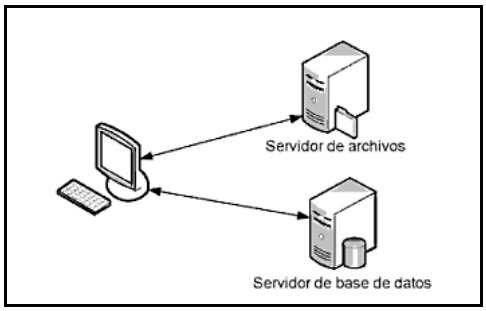


Figura 4. Arquitectura Cliente Servidor

2.1.2 Arquitectura de tres niveles (N-Tier)

En una arquitectura de tres capas el cliente se libera del procesamiento de la lógica de negocio y acceso a datos y se convierte en un cliente ligero.

Una aplicación que sigue este modelo está dividida en las siguientes capas:

Nivel de presentación: Suele consistir en una aplicación cliente que únicamente se encarga de implementar la interface con el usuario. Este nivel en un inicio se implementaba como una aplicación Windows, pero ha ido evolucionando de tal forma que actualmente puede ser también una aplicación web.

Nivel de componentes de aplicación: Son componentes relacionados entre sí que se encargan del procesamiento de la lógica de negocio El nivel de presentación únicamente se comunica con el nivel de negocio para ejecutar las acciones requeridas por el usuario. Normalmente el nivel de negocio está situado en un servidor de componentes de negocio (o varios servidores balanceados) Dentro del servidor de componentes, podemos tener varias capas de componentes (Capa fachada de componentes de negocio, capa de componentes base de negocio, capa de componentes de acceso a datos, etc.)

Nivel de datos: Son normalmente los servidores de bases de datos, como servidores SQL Server, Oracle. DB2, etc. Pero realmente puede ser cualquier fuente de dalos.

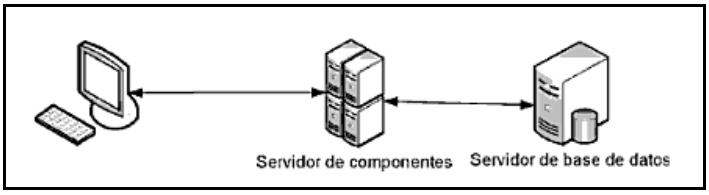


Figura 5. Ejemplo de arquitectura de tres capas

2.1.3 Ventajas de los Sistemas Distribuidos

Una de Las principales características de los sistemas distribuidos es la escalabilidad, Cada capa de la aplicación puede contener tantos servidores balanceados entre si como sea necesario.

Al estar las capas separadas en servidores se aumenta la concurrencia y agilidad de las aplicaciones dando una respuesta más rápida al cliente.

Se aumenta la disponibilidad de la aplicación. La caída de uno de los servidores suele estar respaldado por otro que suple al anterior en caso de fallo. De esta forma tenemos sistemas altamente disponibles.

Aumento de la reutilización de componentes.

2.1.4 Desventaja de los Sistemas Distribuidos

Los costos iníciales son más altos. El sistema está compuesto por más de un servidor con lo cual el costo de la puesta en producción es mayor que el de las aplicaciones monolíticas. Pero a la larga (con sistemas en producción durante varios años y evolucionando los sistemas), el costo total de propiedad es probablemente más bajo que el de las aplicaciones monolíticas.

La administración de las aplicaciones suele ser más costosa debido al carácter distribuido del sistema. Se requieren más conocimientos para poder administrar correctamente una aplicación distribuida.

Se tiene una dependencia muy alta con las redes de comunicación. Un mal funcionamiento de la red afectará negativamente en el rendimiento del sistema distribuido.

Se debe hacer un mayor hincapié en la seguridad de la información. La distribución de los datos da lugar a problemas potenciales de seguridad que hay que tener en cuenta y mitigar en lo posible.

[PRE 05]

2.2 ¿Qué es un Servicio?

Un servicio es simplemente un programa con el que otros programas interactúan mediante mensajes. Un conjunto de servicio instalados/desplegados seria un sistema. Los servicio individuales se deben de construir de una forma consistente (disponibilidad y estabilidad son cruciales en un servicio). Un sistema agregado/compuesto por varios servicios se debe construir de forma que permita el cambio, el sistema debe adaptarse a la presencia de nuevos servicios que aparezcan a lo largo del tiempo después de que se hubieran desplegado ó instalado los servicios y clientes originales. Y dichos cambios no deben romper la funcionalidad del sistema.

Así pues, un servicio es una unidad de trabajo bien definida, que acepta mensajes de entrada y produce mensajes de salida. La funcionalidad del servicio es expuesta al exterior mediante una interface pública bien definida, llamada 'contrato'.

Otro aspecto a destacar es que un servicio-SOA debe de ser, por regla general, interoperable. Para eso, debe de basarse en especificaciones estándar a nivel de protocolos, formato de datos serializado en las comunicaciones, etc. Por eso, el formato de los mensajes suele estar basado en XML, el protocolo de transporte en HTTP y la implementación de dichos servicios basada en estándares como SOAP ó REST.

Sin embargo, cabe la posibilidad de poder desarrollar servicios no interoperables con otras tecnologías, si lo único que se persigue son capacidades de rendimiento.

2.3 Pilares de la Orientación a Servicios

El desarrollo orientado a servicios está basado en los siguientes cuatro pilares, denominados comúnmente en ingles como "Service Orientation Tenets":

2.3.1 Frontera de los Servicios

Una aplicación orientada a servicios a menudo está compuesta por varios servicios distribuidos en diferentes puntos geográficos distantes, múltiples autoridades de confianza y diferentes entornos de ejecución. El costo de traspasar dichas fronteras no es trivial en términos de complejidad y especialmente de rendimiento (la latencia existente en cualquier comunicación remota siempre tiene un costo si el formato de los mensajes es XML-SOAP y el protocolo es HTTP, este "costo" en rendimiento es aun mayor).

Los diseños SOA reconocen estos costos recalcando que hay un "costo" en el momento de cruzar dichas fronteras, por lo que lógicamente, este hecho debe minimizarse en la medida de lo posible.

Debido a que cada comunicación que cruce dichas fronteras tiene un costo potencial, la orientación a servicios se basa en un modelo de intercambio de mensajes explicito en lugar de un sistema de invocación remota de métodos de forma implícita.

Aunque SOA soporta la notación "estilo-RPC" (invocación síncrona de métodos), también puede soportar comunicación asíncrona de mensajes y al mismo tiempo asegurar el orden de llegada de dichos mensajes asíncronos, y poder indicar de forma explícita a qué cadena de mensajes pertenece un mensaje en particular Esta indicación explícita es útil para correlaciones de mensajes y para implementar modelos de concurrencia.

El concepto de que las "fronteras son explicitas" se aplica no solamente a la comunicación entre diferentes servicios, también incluso a la comunicación entre desarrolladores como personas. Incluso en escenarios en los que los servicios se despliegan en un único punto, puede ser común que los desarrolladores del mismo sistema estén situados en diferentes situaciones geográficas, culturales y/o con fronteras organizacionales. Cada una de dichas fronteras incrementa el costo de comunicación entre los desarrolladores. La "Orientación a Servicios" se adapta a este modelo de "desarrollo distribuido" reduciendo el número y complejidad de abstracciones que deban ser compartidas por los desarrolladores a lo largo de las

fronteras de servicios. Si mantenernos el "área de superficie" de un servicio tan pequeña como sea posible, la interacción y comunicación entre las organizaciones involucradas en el desarrollo, se reduce.

Un aspecto que es importante en los diseños orientados a servicios es que la simplicidad y generalización no son un "lujo" sino más bien una aspecto critico de "supervivencia".

Por último y relacionado con la importancia de tener muy en cuenta a "las fronteras"' la idea que se puedes tomar una interface, de un objeto local extenderlo a lo largo de fronteras de diferentes maquinas remotas creando una transparencia en la localización es falsa y en muchos casos dañina. Aunque es cierto que tanto los objetos remotos corno los objetos locales tienen la misma interface desde la perspectiva del proceso que lo consume, el comportamiento de la interface llamada es muy diferente dependiendo de la localización. Desde la perspectiva del cliente, una implementación remota de interface, está sujeta a latencia de red, fallos de red y fallos de sistemas distribuidos que no existen en implementaciones locales. Por todo esto, se debe de implementar una cantidad significativa de código de detección de errores, corrección de lógica para anticiparse a los impactos derivados del uso de interfaces remotas.

2.3.2 Servicios autónomos

Los servicios lógicamente pueden tener dependencias unos de otros en el momento de ejecutarse, sin embargo, lo importante a entender es que deben tener autonomía en "tiempo de desarrollo", es decir, disponer de un área de desarrollo independiente, así como un versionado y despliegue independiente (tanto de código como de base de datos).

Los programas orientados a objetos y aplicaciones cliente servidor e incluso aplicaciones N-Tier no-SOA, normalmente se desplegan/instalan como un único sistema, porque puede ser muy complicado poder hacer despliegues independientes de diferentes módulos/partes de un sistema (por las inter-dependencias que pueda haber con cada modulo). Por todo esto y unido a las complejidades de versionados de interfaces en la orientación a objetos, muchas organizaciones se han vuelto muy conservadoras en como despliegan las aplicaciones.

En la orientación a servicios, la instalación atómica de una aplicación (una única instalación) es realmente la excepción, no la regla. Mientras los servicios individuales se instalan normalmente deforma atómica, los despliegues/instalaciones agregadas de

la mayoría de sistemas y aplicaciones raramente son únicos. Por ejemplo, es muy normal que un servicio individual sea instalado mucho tiempo ames de que una aplicación que lo consuma sea ni siquiera desarrollada y posteriormente desplegada.

También es común en la topología de aplicaciones orientadas a servicios, que los sistemas y servicios evolucionen a lo largo del tiempo, algunas veces sin intención directa de un administrador o desarrollador. El grado en el cual se pueden introducir nuevos servicios en un sistema orientado a servicios depende tanto de la complejidad de las interacciones de los servicios como de la facilidad que ofrezcan los servicios para ser encontrados y explorados. La orientación a servicios recomienda un modelo que incremente las posibilidades de ser encontrado y explorado (por ejemplo mediante UDDI y WSDL), reduciendo la complejidad de las interacciones con los servicios.

El concepto de servicios autónomos también tiene que ver en la forma en que se gestionen las excepciones de los errores. Los objetos se despliegan para ejecutarse en el mismo contexto de ejecución que la aplicación que los consume. Sin embargo. los diseños orientados a servicios asumen que esa situación es una excepción, no la regla. Por esa razón, los servicios esperan que la aplicación que los consume (aplicación cliente) pueda fallar sin darse cuenta el servicio y a menudo sin notificarlo. Para mantener la integridad del sistema, los diseños orientados a servicios hacen uso de técnicas como transacciones, colas persistentes, despliegues redundantes y clúster. Debido a que muchos servicios se despliegan para que funcionen en redes públicas (como Internet) SOA asume que no solamente los mensajes que lleguen pueden estar mal-formados sino que también pueden haber sido modificados transmitidos con malos propósitos. Por eso, los aspectos de seguridad (cifrado, firma, autenticación. autorización, etc.) son críticos en los servicios de hecho la seguridad es de los estándares WS-* más importantes (WS-Security. etc.). La seguridad también puede ser independiente a nivel de servicio.

2.3.3 Esquemas y contratos

La programación orientada a objetos recomienda a los desarrolladores el crear nuevas abstracciones en forma de clases. La mayoría de los entornos de desarrollo modernos no solamente facilitan el definir nuevas clases, sino que los IDEs modernos incluso guían al programador en el proceso de desarrollo según aumenta el número de clases. Las clases son abstracciones convenientes porque comparten estructura y comportamiento en una única unidad específica. SOA sin embargo no recomienda que

los servicios interactúen así, En lugar de esa forma (con tipos. clases, ele.), los servicios interaccionan basándose solamente en esquemas (para estructuras de datos), contratos (para comportamientos) Cada servicio muestra un contrato que describe la estructura de mensajes que puede mandar y/o recibir así como algunos grados de restricciones para asegurar el orden en los mensajes. etc. Esta separación estricta entre estructuras de datos y comportamientos clarifica mucho el desarrollo orientado a servicios.

Debido a que el contrato y el esquema de un servicio dado son visibles durante largos períodos de tiempo y espacio. SOA requiere que los contratos y esquemas se mantengan estables a lo largo del tiempo. Por ejemplo, en Internet sería imposible propagar cambios en un esquema y/o contrato a todas las partes que han consumido alguna vez un servicio. Por esa razón, el contrato y esquema utilizados en diseños SOA tienden a ofrecer más flexibilidad que los interfaces tradicionales orientados a objetos, extendiéndose o ampliándose en lugar de cambiándose interfaces existentes.

2.3.4 Compatibilidad de los servicios

Cada servicio debe de anunciar sus capacidades y exigir sus requerimientos en forma de políticas legibles para el sistema. Una política es al fin y al cabo "algo que se requiere y que se configura de forma declarativa". Por ejemplo que configuraciones y requerimientos de calidad de servicio deben de soportarse para permitir que se consuma un determinado servicio.

Las políticas, recuerdan mucho a la "Orientación a Aspectos", es decir, implementación de "aspectos" de forma declarativa en ficheros de configuración de los programas. AOP (Aspect Oriented Programming) es otro paradigma, que aunque es complementario con la orientación a objetos y con SOA es un tema que no será tratado en este trabajo.

Viendo estos principios nos damos cuenta que la base de la arquitectura orientada a servicios es la autonomía de estos servicios. Los servicios están desacoplados los unos de los otros, no hay dependencia tan cercana como en la programación orientada a objetos.

Debido a que la gran diversidad de entornos donde podrían usarse nuestros servicios, se debe tender al uso de estándares que faciliten la integración. Así mismo no se debe nunca confiar en que los mensajes de entrada que estarnos recibiendo son válidos, debemos siempre realizar comprobaciones muy estrictas de la información que se recibe.

2.4 Conceptos claves de Servicios Web

La mayoría de las aplicaciones orientadas a servicios están implementadas mediante servicios web, pero esto no es realmente obligatorio. Una aplicación orientada a servicios puede implementarse mediante cualquier tecnología que permite el desarrollo de servicios, por ejemplo la cola de mensajería de MSMQ (para Microsoft), JMS (para Java).

A continuación vamos a ver una serie de estándares que forman parte de las arquitecturas orientadas a servicios desarrolladas con servicios web:

2.4.1 WSDL

WSDL son las siglas de *Web Service Description Language*. Es un documento con formato XML que se usa para describir la interface pública de un servicio web así como su localización.

Los principales elementos empleados para construir un documento WSDL son los siguientes:

Elemento	Definición
<portType>	Listado de operaciones ejecutadas por el web service.
<message>	Tipos de mensajes usados por el web service.
<types>	Tipos de datos usados por el web service.
<binding>	Protocolos de comunicación usados por el web service.

Tabla 1. Elementos principales del WSDL

2.4.2 SOAP

SOAP son las siglas de Simple Object Access Protocol. Es un protocolo ligero basado en XML que permite que dos procesos en entornos distribuidos se comuniquen entre si mediante el intercambio de mensajes.

Debido a que estos mensajes son XML, facilitan la lectura, pero tienen la desventaja que suelen ser bastante más pesados que si estos fueran binarios.

SOAP puede ser usado con una gran variedad de protocolos pero se suele usar la mayoría de las veces con HTTP.

La ventaja de SOAP radica en que es un protocolo aceptado por las grandes compañías de software mundiales (Microsoft, IBM, SUN y SAP entre otras), aunque ha otros estándares de servicios también bastante populares para servicios interoperables sencillos y ligeros como REST.

2.4.3 Mensaje SOAP

Como decíamos antes SOAP se basa en el intercambio de mensajes. Un mensaje SOAP está formado por un sobre (envelope) que a su vez contiene un cuerpo y una cabecera.

El sobre (envelope) es un elemento obligatorio. Es el elemento raíz de un mensaje SOAP Define un documento XML como un mensaje SOAP.

La cabecera (header) es un elemento opcional dentro de un mensaje SOAP y es un mecanismo utilizado para entender un mensaje. Por ejemplo podemos utilizar las cabeceras SOAP para añadir información de transaccionalidad o de autentificación en el mensaje.

En el caso que usemos la cabecera en un mensaje SOAP, esta debe ser el primer elemento dentro del sobre (envelope).

El cuerpo (body) es un elemento obligatorio y contiene el mensaje enviado al último destinatario de este.

```
<?xml version="1.0"?>
<SOAP:Envelope
   xmlns:SOAP=http://schemas.xmlsoap.org/soap/envelope/>
   <SOAP:Header>
  <! --Información opcional de cabecera -->
   </SOAP:Header>
    <SOAP:Body>
                  <!--Mensaje va aquí -->
    </SOAP:Body>
</SOAP:Envelope>
```

Figura 6. Extracto mensaje SOAP

Hemos visto que SOAP suele estar relacionado con otros protocolos de transporte como por ejemplo HTTP. Por ello si trazamos un mensaje SOAP nos encontraremos con la siguiente información:

```
POST /item HTTP/1.1
Host: 192.168.1.1
Content-Type: application/soap+xml
Content-Length: 200
```

Figura 7. Traza de mensaje SOAP

Como vemos en la figura 7, las parles principales que conformarán el mensaje son el servidor destinatario de la petición en este caso es una dirección IP, el Content-Type del mensaje y el número de caracteres que va a contener el cuerpo del mensaje.

Si el mensaje es una respuesta contendrá algo parecido a lo siguiente:

```
200 OK
Content-Type: text/plain
Content-Length: 200
```

Figura 8. Extracto de respuesta SOAP

En este caso los datos recibidos contendrán un código estándar HTTP que nos va a indicar si el destinatario ha recibido correctamente el mensaje (como es este caso, 200 OK) o si por el contrario se ha producido un error en la transmisión, también va a contener el tipo de contenido del mensaje así como el número de caracteres.

Para que quede todo más claro a continuación se muestra un ejemplo completo de lo que sería una conversación mediante mensajes SOAP entre dos procesos:

```
POST /articulos HTTP/1.1
Host: www.tienda.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: xxx
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

   <soap:Body xmlns:m="http://www.tienda.com/articulos">
      <m:ObtenerListadoArticulos>
         <m:tipo>portatiles</m:tipo>
      </m: ObtenerListadoArticulos >
   </soap:Body>

</soap:Envelope>
```

Figura 9. Ejemplo de petición SOAP

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: xxx
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
   <soap:Body xmlns:m=" http://www.tienda.com/articulos">
      <m: ObtenerListadoArticulos Response>
         …
      </m: ObtenerListadoArticulos Response>
   </soap:Body>
</soap:Envelope>
```

Figura 10. Ejemplo de respuesta SOAP

2.4.4 UDDI

UDDI son las siglas de Universal Description, Discovery and Integration, es una iniciativa abierta y soportada por OASIS. Es un catálogo independiente basado en XML cuyo propósito es el de facilitar el descubrimiento de servicios expuestos por diversas compañías de forma dinar nica. UDDI provee un mecanismo para navegar y descubrir Servicios Web e interactuar con ellos de manera interactiva o programática.

UDDI es relativamente ligero ya que se ha diseñado solamente como registro y no como repositorio, redirigiendo al consumidor al recurso correcto.

2.5 Integración de Aplicaciones Empresariales

La integración de aplicaciones empresariales es una necesidad de negocio. Todas las empresas tienen una gran diversidad de aplicaciones desarrolladas por distintos fabricantes de software que no tienen capacidad de integración automática entre ellas.

La Integración de las Aplicaciones Empresariales (EAI por sus siglas en ingles) no es otra cosa que la gestión de mensajes, transformación y enrutamiento para que todas estas aplicaciones se puedan comunicar entre sí.

Actualmente existen dos tipos de arquitecturas básicas para solucionar este problema. Una de ellas es la arquitectura centralizada y otra es una arquitectura en bus.

[MIC 08]

2.5.1 Integración centralizada

En una integración centralizada vamos a tener un concentrador (hub) el cual se encargará de conectar servicios descentralizados entre sí.

Para que el concentrador se pueda comunicar con los servicios vamos a usar la figura de los adaptadores, los cuales se encargaran de realizar la conexión con las aplicaciones que proporcionan los servicios deseados, convertir el formato de los mensajes proporcionados por estos servicios en un formato que el concentrador entienda y viceversa.

Teniendo un solo concentrador hace que la integración sea mucho más sencilla debido a que si queremos integrar nuestro sistema con otros servicios solo tenemos que añadir esos servicios al concentrador. Esta arquitectura es mucho más viable y efectiva que la integración punto a punto que realizan algunas empresas.

La integración punto a punto consiste en que cada una de las aplicaciones o servicios que queramos integrar se comuniquen unas con otras sin que haya ningún sistema que intermedie.

El problema de esta "no metodología" es que el resultado es una integración muy difícil de mantener. Cada nuevo servicio que queramos integrar hará que el sistema crezca exponencialmente. Para integrar 3 servicios entre si necesitaremos 3 conexiones para completar la integración, pero si necesitáramos conectar 5 servicios ya necesitaríamos 10 conexiones y finalmente supongamos que quisiéramos integrar 10 servicios necesitaríamos 45 conexiones punto a punto.

Vemos que el crecimiento exponencial hace que esta aproximación a la integración de aplicaciones no sea el idóneo.

Uno de los inconvenientes que tiene la integración centralizada es la escalabilidad. Al estar toda la integración de servicios centralizada en un único punto el único modo de escalar el sistema es aumentando el hardware y muchas veces como sabemos no es suficiente para que nuestro sistema tenga un rendimiento aceptable.

2.5.2 Entreprise Service Bus

La arquitectura basada en bus usa un canal central para la propagación de mensajes Las aplicaciones van a proporcionar información al bus mediante el uso de adaptadores que les faciliten esa tarea.

El enrutamiento de los mensajes se realiza de forma transparente para el consumidor evitando la necesidad de que este conozca la ubicación del servicio hay otro tipo de enrutamiento basado en el contenido. Este tipo de enrutamiento ejecuta una serie de reglas de negocio basadas en el contenido del mensaje para averiguar quién es el destinatario de este.

El núcleo de Enterprise Service Bus proporciona un método de transporte fiable y distribuido que emplea un mecanismo de almacenamiento y reenvío gracias al cual se garantiza la entrega de los mensajes incluso en caso de un mal funcionamiento de la red.

Enterprise Service Bus no implementa una arquitectura orientada a servicios, sino que proporciona las bases para que pueda ser implementada.

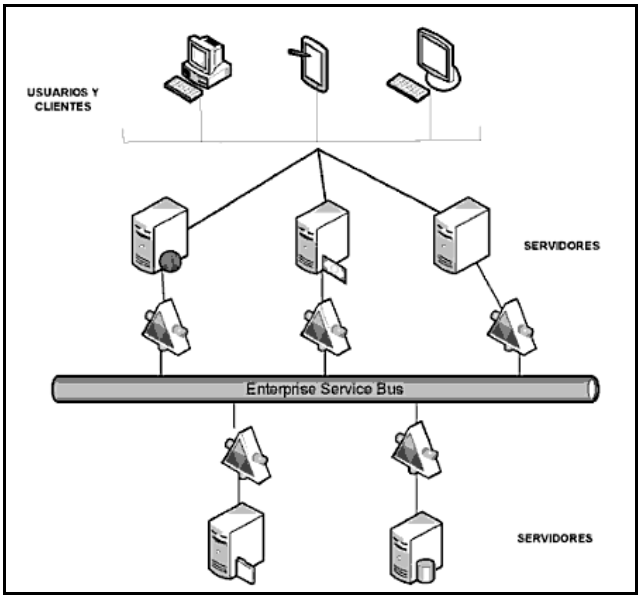


Figura 11. Integración se servicios mediante ESB

2.6 Java Business Integration

Java Business Integration (JBI) es una especificación desarrollada bajo Java Community Process (JCP) con el objetivo de implementar en Java una Enterprise Application Integration (EAI), siguiendo los principios de la Arquitectura Orientada a Servicio (SOA). La referencia JCP fue la JSR 208 para JBI 1.0 y la JSR 312 para JBI 2.0.

2.6.1 Componentes del JBI dentro del ESB

JBI proporciona una arquitectura donde los componentes aparecen en forma de plug-in´s conectados a contenedores JBI y actuando como proveedores de servicio o consumidores de servicio, aunque este rol puede intercambiarse o incluso combinarse. Está construido teniendo en mente los Servicios Web, aunque no son estrictamente necesarios para su funcionamiento. Los componentes, según su función pueden ser de dos tipos:

Service engines

Manejan la lógica de negocio. Pueden implementar procesos de negocio, consumir otros servicios, transformaciones de datos, etc.

Binding components

Se usan para enviar y recibir mensajes a través de transportes y protocolos particulares, que por tanto nada tienen que ver con la especificación de mensajes de JBI. Pretenden aislar al entorno JBI del heterogéneo exterior, normalizando los mensajes entrantes y desnormalizando los mensajes salientes, garantizando que en el interior se trabaje únicamente con mensajes normalizados. Esta funcionalidad permite que JBI pueda alcanzar servicios que no cumplen esta especificación, o al contrario, que sistemas externos puedan consumir servicios JBI de forma transparente.

El modelo usado para describir los servicios, uno de los fundamentos SOA, es Web Services Description Language (WSDL) 2.0. Además se usa el mismo modelo para realizar las invocaciones de servicio. El mecanismo central de gestión de estos mensajes, el Normalized Message Router (NMR), despacha los mensajes normalizados que le llegan, enviándolos al componente adecuado. El hecho de que el NMR actúe como mediador entre los componentes garantiza que el acoplamiento entre ellos sea muy débil, de manera que éstos sólo se preocupan de que el mensaje sea adecuado para el NMR, y por ello simplifican enormemente la interacción entre consumidor y proveedor. La comunicación se realiza mediante uno de los cuatro tipos de patrones de secuencias soportados, los Patrones de Intercambio de Mensajes o Message Exchange Patterns (MEPs):

a. **In-Only**: El patrón estándar de envío en un solo sentido, donde el consumidor envía un mensaje al proveedor que responde únicamente con el estado (normalmente done).

b. **Robust In-Only**: Este patrón consigue una versión más confiable del intercambio en un solo sentido. El consumidor inicia el intercambio con un mensaje al cual el proveedor responde con un estado. Si por el contrario responde con un anuncio de error (un fault), el consumidor debe responder en el estado.

c. **In-Out**: El patrón estándar de intercambio de mensajes, con un mensaje en cada sentido. El consumidor inicia con un mensaje, el proveedor responde con un mensaje respuesta o fallo, y el consumidor termina el intercambio con un estado.

d. **Optional-Out**: Patrón similar al In-out, en el que la respuesta del proveedor es opcional.

Capítulo 3: Estado del Arte

En este capítulo se detallan la forma del análisis y diseño de los distintos tipos paradigmas para el desarrollo de software, luego se realiza la comparación entre dos ESB no licenciados (OpenESB y ServiceMix), todo esto enmarcado para el fututo diseño de la integración de aplicaciones mediante un ESB.

3.1 Deslatech: EAI para Hospitales de Brasil

3.1.1 Problema

Deslatech es un integrador de sistemas en Brasil y debe construir una solución para KitMed, un operador logístico que opera el almacén y farmacia de los múltiples hospitales en Brasil. Esto implica la sincronización de los productos y listas de precios y la transferencia de las órdenes de compra y facturas electrónica a través de varios centros distribuidos geográficamente.

Para este fin busca herramientas que puedan suministrar todas las características de interoperabilidad en un escenario de EAI y B2B para hospitales y almacenes farmacéuticos, ya sea como un centro de comunicación con el almacén o como una plataforma de integración entre el ERP del hospital y los sistemas de gestión.

3.1.2 Propuesta de solución

Se está manejando los procesos críticos que involucran la vida humana, como consumibles médicos y el suministro de medicamentos del almacén, por lo que es necesario tener una plataforma tecnológica fiable y exenta de posibles errores.

Se inicio analizando las distintas soluciones en el mercado para EAI para realizar B2B y al analizar que GlassFish se podía utilizar como un centro de comunicación (integracion) entre los ERP de los hospitales y los sistemas de gestión. La especificación JBI, la flexibilidad de desarrollar más de un contenedor EJB y la prueba de carga que se hizo dio como resultado la robustez requerida.

El proceso de evaluación contó con un corto tiempo para elegir la herramienta para ser utilizada. Se siguió un enfoque pragmático centrado en las necesidades funcionales y añadiendo algunas pruebas de carga y las limitaciones de rendimiento. Los requisitos más importantes fueron:

- Mapping con texto plano, validación y transformación.
- Soporte para transacciones distribuidas (Oracle / SQL Server / Web Services).
- Proceso y modelado de aplicaciones compuestas para manejar negocios dinámicos.
- Permitir el monitoreo de los procesos y entornos.
- Reducción de las intervenciones a nivel de código para los mantenimientos.
- Estabilidad en las instancias del software.

Todas las pruebas se realizaron con datos reales y todo el proceso de desarrollo es muy lineal y con ajustes de código muy bajo. Cada aspecto de las aplicaciones esta a cargo de NetBeans, lo que simplifica las pruebas y despliegue.

No es una aplicación tradicional de EAI que requiere transformación de mensajes y procesos de reglas de negocio dinámicas y complejas. Deslatech escogió GlassFish debido a la interoperabilidad y la flexibilidad que ofrece el contenedor.

La solución se basa en OpenESB y OpenMQ que se utiliza para controlar los cuellos de botella en el procesamiento y en fallos transitorios. GlassFish y NetBeans integrados les ayudó con su tiempo de salida al mercado. El uso de diferentes aplicaciones de Oracle, SQL Server e Informix como back-ends de base de datos.

3.1.3 Plataforma tecnológica

Servidor de aplicaciones: Sun Java System Application Server (GlassFish) 9.1_02.

El ambiente de desarrollo se ejecuta en Windows XP, Ubuntu Linux y Red Hat.

El ambiente para el control de calidad y producción se está ejecutando sobre Linux Red Hat y el hardware es un poco heterogéneo.

Los módulos y características de GlassFish utilizadas en el proyecto son:

La integración externa (entre el centro de distribución y hospitales), que involucra la manipulación mensajes, transformación dinámica y reglas de negocio complejas todo sobre la base de JBI. Los elementos que intervienen son componentes vinculantes como archivo, ftp y SOAP. Web Services se implementan en el contenedor EJB para los sistemas de legado, utilizando recursos como el JDBC o colas JMS. También se construye algunas aplicaciones web para proporcionar información y seguimiento para las integraciones de aplicaciones.

La integración interna, es decir, lo que sucede dentro del hospital, donde se produce la comunicación entre los sistemas de gestión de farmacia almacén, ERP del hospital y el sistema de gestión de la salud son proporcionados por las aplicaciones empresariales, tomando ventajas de rendimiento, la escalabilidad y la coherencia transaccional.

Algunas rutinas están utilizando colas en OpenMQ con el fin de manejar los cuellos de botella en el procesamiento de los pedidos al software y los errores transitorios, ya que se tiene muchos sistemas, tecnologías y entornos diferentes.

Se utilizando Open ESB, incluido en el servidor GlassFish, como proveedor de JBI. La persistencia es resuelta por el JPA construido encima de TopLink. Para el proceso de desarrollo se utiliza NetBeans y Subversion como controlador de versiones. Las tareas programadas (Jobs) se fabricaron a partir de componentes Quartz.

3.2 CESCE apunta hacia SOA

3.2.1 Problema

En el año 1970 nace CESCE, sociedad anónima participada mayoritariamente por el Estado Español y por los principales bancos y empresas aseguradoras del país, con la finalidad de asegurar a las distintas empresas de los posibles riesgos de impago derivados de las ventas de sus productos y servicios tanto en el mercado interior como exterior.

Con el paso del tiempo, las necesidades de la empresa respecto a sus aplicaciones de gestión han crecido y se han tornado más exigentes en calidad, complejidad y urgencia, por lo que se plantearon una serie de retos a llevar a cabo necesariamente por parte del área TI: Dar rápida respuesta a las necesidades del negocio, reducir los tiempos de toma de decisiones y de respuesta, simplificar en número y complejidad las aplicaciones a utilizar, reducir los costes de mantenimiento, unificar y estandarizar y, por último, reducir al mínimo la intervención humana en los procesos controlándolos al máximo.

Mariano Arnáiz Mateo, director de Informática y Operaciones en CESCE, resumía así la actuación de la empresa: "Ante estas necesidades, desde el Área de TI decidimos evolucionar nuestra arquitectura a un modelo que le permita responder de forma más ágil a las necesidades de negocio, minimizando los esfuerzos e inversiones en el desarrollo de software". Es decir, avanzando hacia una arquitectura SOA en la que encontrar una reducción de los tiempos de desarrollo y puesta en producción, la

reutilización de los desarrollos ya existentes y consolidados, adaptándolos a la nueva arquitectura, una gran evolución hacia estándares tecnológicos y hacia la automatización de procesos.

Mariano Arnáiz Mateo, director de Informática y Operaciones en CESCE, resumía así la actuación de la empresa: "Ante estas necesidades, desde el Área de TI decidimos evolucionar nuestra arquitectura a un modelo que le permita responder de forma más ágil a las necesidades de negocio, minimizando los esfuerzos e inversiones en el desarrollo de software". Es decir, avanzando hacia una arquitectura SOA en la que encontrar una reducción de los tiempos de desarrollo y puesta en producción, la reutilización de los desarrollos ya existentes y consolidados, adaptándolos a la nueva arquitectura, una gran evolución hacia estándares tecnológicos y hacia la automatización de procesos.

3.2.2 Propuesta de solución

El primer paso que en CESCE se ha dado hacia SOA, se ha hecho junto a Software AG y su herramienta Crossvision, y se concretó en el sistema de gestión de Call Center, cuyo escenario demandaba bastantes cambios ya que carecía de canales de entrada al Contact Center unificados, usaba hasta tres sistemas de gestión distintos para una única transacción, la navegación no estaba optimizada y, además, el intercambio de datos era manual e imposible de adaptar a nuevas necesidades. La solución, según Arnáiz, "es una Interfaz de Usuario Única, integrada con todos los sistemas, para que los usuarios del call center, obtengan la información de la forma que lo necesitan: unificada, por lo que la aplicación es mucho más intuitiva y redunda en una mayor productividad".

En cuanto a las ventajas de SOA, Arnáiz señala que "antes se compraba una pieza y al realizar cambios había que de-secharla completamente, ahora se trata como de un 'Lego' en el que se ensamblan y añaden las piezas para cubrir nuevas necesidades de negocio sin tener que volver a empezar de cero ante un cambio. Nuestra infraestructura de TI se va a simplificar, por una sencilla razón, porque empiezo a reutilizar los elementos comunes. Esto aporta mayor agilidad y, sobre todo, una reducción de los costes".

Tres meses ha sido el tiempo empleado para implantar la nueva solución tras seleccionar el proyecto. La elegida fue Software AG, ya que disponía de una solución completa y modular con la que no cubrir una necesidad puntual, sino poder diseñar

una estrategia de implantación de SOA a largo plazo, además de por su conocimiento del negocio de CESCE, y de su arquitectura Legacy basada en esta tecnología. Ampliar el proyecto de Interfaz Único a las demás áreas es el siguiente paso, que tendrá que incorporar nuevas piezas como la plataforma de BPM. Además, será necesario integrar todas las piezas de la plataforma tecnológica dentro de la nueva arquitectura y establecer una metodología orientada a SOA.

3.2.3 Plataforma tecnológica

Basada al 100% en arquitectura SOA.

- Desarrollo de una nueva interfaz de usuario, utilizando Crossvision Application Componer.
- Integración de la interfaz de usuario con los sistemas Legacy basada en Web Services:
- Reutilización de Web Services ya existentes.
- Desarrollo de nuevos Web Services mediante Crossvision Legacy Integrator.
- "Empaquetado" de navegaciones sobre el emulador del HOST, mediante Crossvision Legacy Integrator.
- Combinación de Web Services, para crear otros más complejos, mediante Crossvision Service Orchestrator.
- Desarrollo de una nueva interfaz de usuario, utilizando Crossvision Aplication Designer.
- Integración de la Interfaz de Usuario con los sistemas Legacy basada en Web Services:
- Reutilización de Web Services ya existentes.
- Desarrollo de nuevos Web Services mediante CV Legacy Integrator (EntireX).
- "Empaquetado" de navegaciones sobre el emulador del HOST, mediante Crossvision Legacy Integrator (ApplinX).
- Combinación/Invocación de Web Services, para crear otros más complejos, mediante Crossvision Service Orchestrator.

Capítulo 4: Aporte teórico

En este capítulo se detallan la forma del análisis y diseño de los distintos tipos paradigmas para el desarrollo de software, luego se realiza la comparación entre dos ESB no licenciados (OpenESB y ServiceMix), todo esto enmarcado para el fututo diseño de la integración de aplicaciones mediante un ESB.

Finalmente se realiza la adaptación del SOAD para que sea aplicable a la solución del problema.

4.1 Análisis y Diseño Orientado a Objetos

En el software hay varias formas de enfocar un modelo. Las dos formas más comunes son la perspectiva algorítmica y la perspectiva orientada a objetos.

La visión tradicional del desarrollo de software toma una perspectiva algorítmica. En este enfoque, el bloque principal de construcción de todo el software es el procedimiento o función. Esta visión lleva a los desarrolladores a centrarse en cuestiones de control y de descomposición de algoritmos grandes en otros más pequeños. No hay nada inherentemente malo en este punto de vista, salvo que tiende a producir sistemas frágiles. Cuando los requisitos cambian y el sistema crece, los sistemas construidos con un enfoque algorítmico resultan muy difíciles de mantener.

La visión actual del desarrollo de software toma una perspectiva orientada a objetos. En este enfoque, el principal bloque de construcción de todos los sistemas software es el objeto o clase. Para explicarlo sencillamente, un objeto es una cosa, generalmente extraída del vocabulario del espacio del problema o del espacio de la solución. Una clase es una descripción de un conjunto de objetos que son lo suficientemente similares (desde la perspectiva del modelador) para compartir una especificación. Todo objeto tiene una identidad (puede nombrase o distinguirse de otra manera de otros objetos), estado (generalmente hay algunos datos asociados a él) y comportamiento (se le pueden hacer cosas al objeto, y él a su vez puede hacer cosas a otros objetos).

Por ejemplo, considérese una arquitectura sencilla de tres capas para un sistema de contabilidad, que involucre una interfaz de usuario, un conjunto de servicios del negocio y una base de datos. En la interfaz del usuario aparecerán objetos concretos tales como botones, menús y cuadros de dialogo. En la base de datos aparecerán objetos concretos, como tablas que representaran entidades del dominio del problema,

incluyendo clientes, productos y pedidos. En la capa intermedia aparecerán objetos tales como transacciones y reglas de negocio, así como vistas de más alto nivel de las entidades del problema, tales como clientes, productos y pedidos.

Actualmente, el enfoque orientado a objetos forma parte de la tendencia principal para el desarrollo de software, simplemente porque a demostrado ser valido en la construcción de sistemas de toda clase de dominios de problemas abarcando todo el abanico de tamaños y complejidades. Más aun, la mayoría de lenguajes actuales, sistemas operativos y herramientas son orientados a objetos de alguna manera, lo que ofrece más motivos para ver el mundo en términos de objetos. El desarrollo orientado a objetos proporciona la base fundamental para ensamblar sistemas a partir de componentes utilizando tecnologías como J2EE o .NET.

Hablar del análisis y diseño orientado a objetos, es hablar del Lenguaje de Marcado Universal, UML por sus siglas en ingles, el cual es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema con gran cantidad de software.

4.1.1 Documentación

Una organización de software que trabaje bien produce toda clase de artefactos además de código ejecutable. Estos artefactos incluyen (aunque no se limitan a):

- Requisitos.
- Arquitectura.
- Diseño.
- Código fuente.
- Planificación de proyectos.
- Pruebas.
- Prototipos.
- Versiones.

Dependiendo de la cultura de desarrollo, alguno de estos artefactos s tratan más o menos formalmente que otros. Tales artefactos no son tan solo los entregables de un proyecto, también son críticos para el control, la medición y comunicación que requiere un sistema durante su desarrollo y *después de su despliegue*.

UML cubre la documentación de la arquitectura de un sistema y todos sus detalles. UML también proporciona un lenguaje para expresar requisitos y pruebas.

4.1.2 Diagramas en UML

Un diagrama es la representación gráfica de un conjunto de elementos, visualizado la mayoría de las veces como grafo conexo de nodos (elementos) y arcos (relaciones). Los diagramas se dibujan para visualizar un sistema desde diferentes perspectivas, de forma que un diagrama es la proyección de un sistema. En todos los sistemas, excepto en los más triviales, un diagrama representa una vista resumida de los elementos que constituyen un sistema. El mismo elemento puede aparecer en todos los diagramas, solo en unos pocos diagramas (el caso más común), o en ningún diagrama (el caso muy raro). En teoría un diagrama puede contener cualquier combinación de elementos y relaciones. En la práctica, sin embargo, solo surge un pequeño número de combinaciones habituales, las cuales son consistentes con las cinco vistas más útiles para comprender la arquitectura de un sistema con gran cantidad de software. Por esta razón, UML incluye trece tipos de diagramas:

1. Diagrama de clases.
2. Diagrama de objetos.
3. Diagrama de componentes.
4. Diagrama de estructura compuesta.
5. Diagrama de casos de uso.
6. Diagrama de secuencia.
7. Diagrama de comunicación.
8. Diagrama de estados.
9. Diagrama de actividades.
10. Diagrama de despliegue.
11. Diagrama de paquetes.
12. Diagrama de tiempos.
13. Diagrama de visión global de interacciones.

[BOO 08]

4.2 Modelado de Procesos de Negocios

Como un componente integral para la gestión de procesos empresariales (Business Process Management - BPM), el software de modelado es una herramienta empresarial de valor incalculable para desplegar, planificar, probar y cambiar los procesos de negocios antes de su implementación. Proporcionar el vínculo entre las capacidades esperadas del BPM y el estado actual, el modelado BPM puede cuantificar el rendimiento y las previsiones del sistema para responder a los cambios internos y externos.

Los modelos de negocio se introdujeron por primera vez en la década de 1990. Sin conexión con el software BPM en ese momento (que no existía en su versión actual), se desarrollaron con un éxito limitado. Ahora, las herramientas de modelado son un componente integral de software BPM, aunque también siguen existiendo como soluciones independientes.

- El modelado de procesos de negocio ayuda a las organizaciones a:
- Organizar los procesos de negocio utilizando un framework formal y lógico.
- Facilitar el proceso de documentación en un registro único y coherente
- Integrar los modelos de procesos a los datos, sistemas y organizaciones.
- Identificar los cuellos de botella del sistema.
- Relacionar los procesos de negocio a los recursos propios de la organización.

4.2.1 Modelado Práctico

El primer paso práctico en la implementación de BPM es el análisis, modelado y simulación dentro de tres etapas que permite conocer la organización de los procesos y el desempeño de áreas claves del negocio:

Modelado: conecta y caracterización y refleja los procesos de negocio en un modelo del sistema para evaluar el flujo proceso lógico. Posterior a la implementación, el modelado ayuda a cuantificar el impacto de cambios en el proceso.

Análisis: analiza las métricas clave para determinar el rendimiento, establecer a operatividad y la eficiencia del proceso, y determinar el ROI para justificar la implementación del BPM.

Simulación: los datos de prueba se ejecutan para medir el rendimiento del sistema e introducir "what if" (que sucedería en caso de) para medir las respuestas a escenarios cambiantes.

4.2.2 Desarrollar un modelo de visión

La clave para construir un modelo con éxito es centrarse en objetivos de la organización y, a continuación se entrelaza el comportamiento esperado de los recursos, el valor y las cadenas de proveedores y procesos de negocio en toda la organización. En el desarrollo de estrategias para alcanzar las metas organizacionales, los analistas de negocio deben ser conscientes de que la asociación entre las estrategias y procesos es muy importante, y que las cadenas de valor es probable que cruzar varios departamentos o entidades dentro de la organización.

4.2.3 Determinar el proceso

Busca documentar de procesos de negocios, su lógica y el flujo para alcanzar los objetivos de la organización. Por ejemplo, los procesos de autenticación de crédito en una compañía de seguros podrían requerir que el solicitante sea autenticado mediante diversos controles: que el importe del reclamo, el límite de la póliza haya pasado por una comprobación de validez y así sucesivamente.

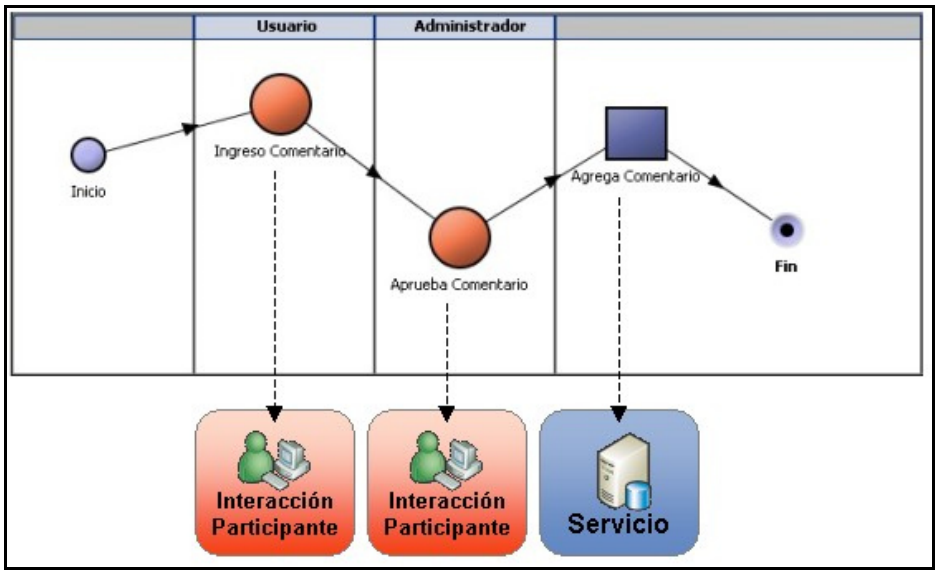


Figura 12. BPM integra en los flujos a los sistemas

4.2.4 Modelando: Cómo ayudaría

La forma en que el software BPM está diseñado hoy en día hace que sea fácil para los usuarios definir reglas de negocio en sus modelos. Cuando se construye sobre las estándares apropiados, los nuevos modelos de negocio trabajar de manera boundary-less(fácil comunicación) con otras aplicaciones empresariales, lo que permite a los usuarios definir sin restricción "bajo demanda" cambios a medida que se necesiten.

Esto permite a los modelos de negocio incorporar datos del mundo real que van desde los cálculos aritméticos simples a escenarios muy complejos que implican las reglas de negocio, fórmulas y cálculos complejos.

Las empresas utilizan modelos para lograr una mayor claridad, mediante la definición de las capacidades clave que se espera del sistema BPM.

4.3 Análisis y Diseño Orientado a Servicios

La metodología de modelado y diseño para aplicaciones SOA se conoce como análisis y diseño orientado a servicios (SOAD, Service Oriented Analysis and Design). La arquitectura orientada a servicios es tanto un marco de trabajo para el desarrollo de software como un marco de trabajo de implementación. Para que un proyecto SOA tenga éxito los desarrolladores de software deben orientarse ellos mismos a esta mentalidad de crear servicios comunes que son orquestados por clientes o middleware para implementar los procesos de negocio. El desarrollo de sistemas usando SOA requiere un compromiso con este modelo en términos de planificación, herramientas e infraestructura.

SOAD, facilita el modelado End-to-End y tiene herramientas de soporte para que sea entendible por el desarrollador. SOA trae flexibilidad y agilidad para los negocios, esto supone métodos que abarcan desde el dominio del negocio, la arquitectura y la aplicación.

Cuando la mayoría de la gente habla de una arquitectura orientada a servicios están hablando de un juego de servicios residentes en Internet o en una intranet, usando servicios web. Existen diversos estándares relacionados a los servicios web. Incluyen los siguientes: XML, HTTP, SOAP, WSDL, UDDI.

4.3.1 Elementos de SOAD

Los siguientes requisitos se han definido para SOAD:

- Proceso y notaciones se deben definir de una manera formal (o al menos semi-formal), al igual que en cualquier otro proyecto o metodología de diseño. SOAD no tiene que empezar de cero. Al seleccionar y combinar el desarrollo Orientado a Objetos, BPM, y los elementos de Aplicaciones Empresariales, los elementos adicionales se puede definir si es necesario.
- Tiene que haber una forma estructurada para conceptualizar servicios:
    - El análisis y diseño orientado a objetos (OOAD) nos da clases y objetos en el nivel de aplicación, mientras que los modelos de procesos BPM dan event-driven. SOAD tiene la función de pegamento para unir todo lo mencionado.
    - El método ya no está orientado a casos de uso, pero está dirigido por los acontecimientos y procesos de negocio. El modelado de casos de uso viene como un segundo paso en un nivel inferior.
    - El método incluye la sintaxis, la semántica y las políticas. Esto es necesario para la composición ad hoc, la intermediación semántica, y el descubrimiento de tiempo de ejecución.
- SOAD debe proporcionar de manera bien definida, los factores de calidad y las mejores prácticas (por ejemplo, responder a la granularidad). Los roles planteados por BPEL deben ser contestadas. Por ejemplo, ¿Quién es responsable de qué porción del trabajo: el Desarrollador, Arquitecto, o analista?
- SOAD también tendrá que responder a la pregunta: ¿Qué no es un buen servicio? Por ejemplo: cualquier cosa que no es o que probablemente no sea reutilizable, probablemente no hace un buen uso de SOA. Otro ejemplo sería si es embebido, sistemas en tiempo real con un propósito, que requisitos no funcionales no pueden permitirse ninguna sobrecarga de procesamiento XML.
- SOAD debe facilitar el modelado de extremo a extremo y contar con el apoyo de herramientas completas. Si SOA se supone que aporta flexibilidad y agilidad para el negocio, lo mismo debería esperarse de su método de apoyo, y en los dominios del negocio, de la arquitectura y el diseño de aplicaciones.

Además de la combinación de desarrollo OO, BPM, y las técnicas de Aplicaciones Empresariales, hay varios conceptos y aspectos importantes SOAD, que todavía tienen que ser complementados por:

- Servicio de categorización y agregación.
- Políticas y aspectos.
- Procesos Meet-in-the-middle (conveniente en el medio).
- Intermediación Semántica.
- Servicio de recolección y de mediación de reconocimiento.

Servicio de categorización y agregación, los servicios tienen diferentes usos y propósitos, por ejemplo, los servicios de software se pueden distinguir de los servicios empresariales. Además, los servicios pueden ser orquestados atómicamente (composición) en un nivel superior.

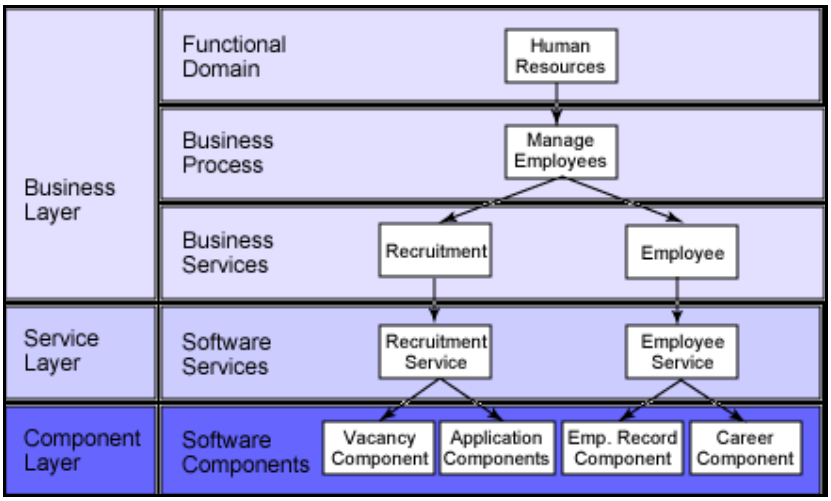


Figura 13. SOAD jerarquía de definición de servicios

Políticas y aspectos, Un servicio tiene una sintaxis, semántica y características QoS características que tienen que ser modeladas; los contratos formales de interfaz tienen que cubrir más que la descripción de servicios Web WSDL (Lenguaje). Por lo tanto, el marco de WS-Policy es una especificación importante.

Procesos Meet-in-the-middle, no hay ningún proyecto que inicie totalmente en blanco en el mundo real como es el caso de las aplicaciones heredadas (aplicación heredada es un sinónimo de las aplicaciones existentes). El enfoque de abajo hacia arriba tiende a conducir a las abstracciones pobres de servicios-negocio en este caso el diseño del entorno de TI existente es quien dirige, en lugar de las necesidades existentes y futuras de negocio. Por otro lado, el procesamiento de arriba hacia abajo puede causar deficiencias en los requisitos no funcionales, y comprometer otros factores de calidad de la arquitectura (por ejemplo, problemas de rendimiento debido a

la falta de normalización en el modelo de dominio), así como desajustes de impedancia en el servicio y el componente de la capa.

Intermediación semántica, en cualquier contexto de SOA, una interfaz de contrato formal para la sintaxis de invocación es importante. El problema de la semántica (el significado de los parámetros, etc.) tiene que ser resuelto (modelado de dominios). Esto es clave en cualquier B2B e invocación de escenarios dinámicos.

Servicio de recolección y de mediación de reconocimiento, los servicios deben ser identificados y definidos con la reutilización como uno de los principales criterios de conducción de la SOA en mente. Si un componente (o servicio) no tiene potencial de reutilización, entonces probablemente no debería ser desplegado como un servicio.

4.3.2 Factores de Calidad

Algunos principios generales o factores de calidad que se pueden identificar como base del diseño dentro de SOAD:

- Servicios bien diseñados aportan flexibilidad y agilidad para el negocio, también facilitan el trabajo de reconfiguración y la reutilización a través de la articulación flexible, encapsulación y ocultación de información.
- Unos servicios bien concebidos son significativos y aplicables a más de aplicación de empresa, las dependencias entre los servicios se reducen al mínimo y explícitamente.
- Los servicios son abstracciones coherente, completa y consistente, por ejemplo, uno debe pensar en crear, leer, actualizar, borrar y buscar (CRUDS) cuando se diseñen estos servicios y posteriormente se asignen.
- Con frecuencia los servicios son sin estado (stateless).
- El servicio de nombres es comprensible para los expertos del dominio sin conocimientos técnicos profundos.
- En una SOA, todos los servicios deben seguir la misma filosofía de diseño (que se articula a través de patrones y plantillas) y los patrones de interacción, el estilo arquitectónico puede ser fácilmente identificado (por ejemplo, durante las revisiones de arquitectura).
- El desarrollo de los servicios y los consumidores de servicios sólo requiere conocimientos básicos del lenguaje de programación, además del conocimiento

del dominio, conocimientos en middleware sólo se requiere estos conocimientos para unos pocos especialistas.

4.3.3 Identificación y Definición de Servicios

De arriba hacia abajo, las técnicas de modelado de nivel empresarial puede servir de punto de partida de las actividades de modelado SOA. Pero como ya se señaló, una implementación de SOA rara vez se inicia en un escenario limpio, la creación de una solución SOA casi siempre conllevan una integración de los sistemas existentes legado por la descomposición de ellos en los servicios, operaciones, procesos de negocio, y reglas de negocio.

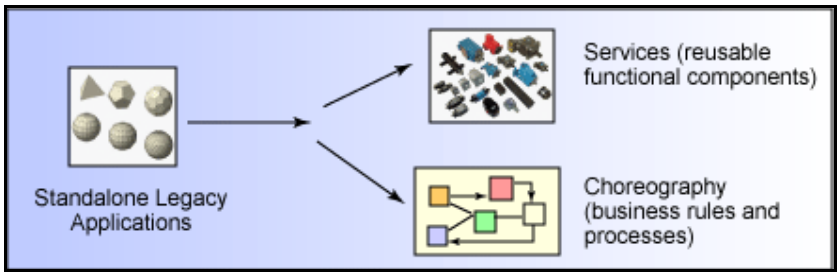


Figura 14. Descomposición de servicios

Todas las técnicas de desarrollo OO se puede aplicar en relación con la identificación y definición de un servicio, pero desde un punto de vista de las necesidades superiores que deben adoptarse.

Análisis del negocio directo e indirecto, BPM y el análisis de requerimientos directo a través de entrevistas a los interesados son una manera obvia y adecuada de l identificación los servicios candidatos. La experiencia demuestra que esta vía principal debe ser modificada por técnicas complementarias indirectas, como las entrevistas a jefes de productos y otros líderes del negocio.

Descomposición del dominio, el análisis del subsistema, la creación de modelo de metas y técnicas relacionadas, son una prometedora primera propuesta para un proceso de estructuración SOA para el método o marco de la conceptualización de servicio. El trabajo FODA también contribuye a este punto.

Granularidad del Servicio, seleccionar el nivel adecuado de abstracción es un elemento clave al modelar servicios. Con frecuencia se escucha el consejo de modelo de grano grueso, esta es una simplificación excesiva y ligera. Se debe reformular el

modelo de grano grueso en lo posible, sin perder o comprometer la relevancia, coherencia e integridad. Hay espacio para abstracciones de servicio de grano fino en cualquier SOA, suponiendo que hay una necesidad de negocio. Como SOA no es igual a los servicios Web y SOAP, diferentes protocolos se puede utilizar para acceder a servicios que residen en los diferentes niveles de abstracción.

Convenciones de nombres, Un esquema de nombres en todas las aplicaciones empresariales (espacios de nombres XML, los nombres de paquetes de Java, dominios de Internet) deben definirse. Un ejemplo sencillo sería recomendar siempre la asignación de un servicio con un nombre, y sus operaciones con los verbos. Esta buena práctica se origina en el espacio de desarrollo OO.

4.4 Ciclo de Vida del Desarrollo de Software en JUNTOS

4.4.1 Arquitectura

A continuación se muestran todas las arquitecturas de las distintas aplicaciones de la institución.

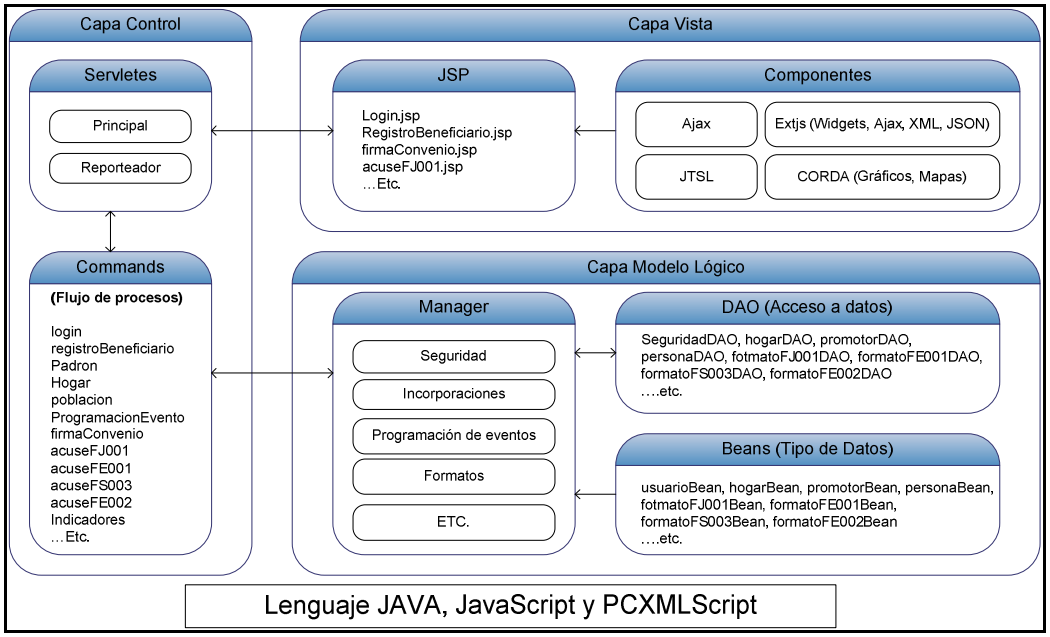


Figura 15. Sistema Web JUNTOS - Arquitectura software

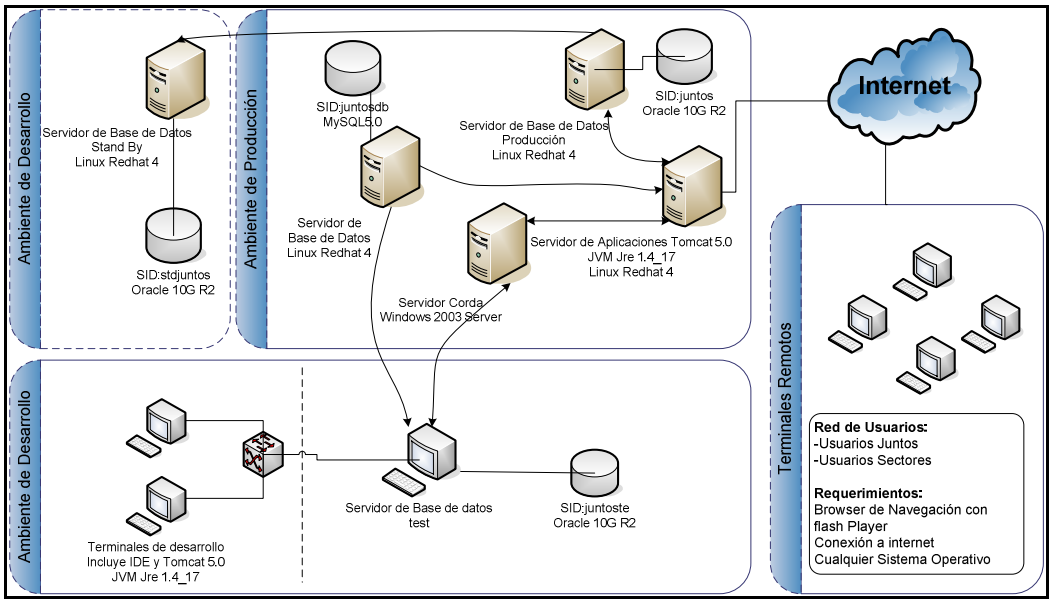


Figura 16. Arquitectura hardware y software

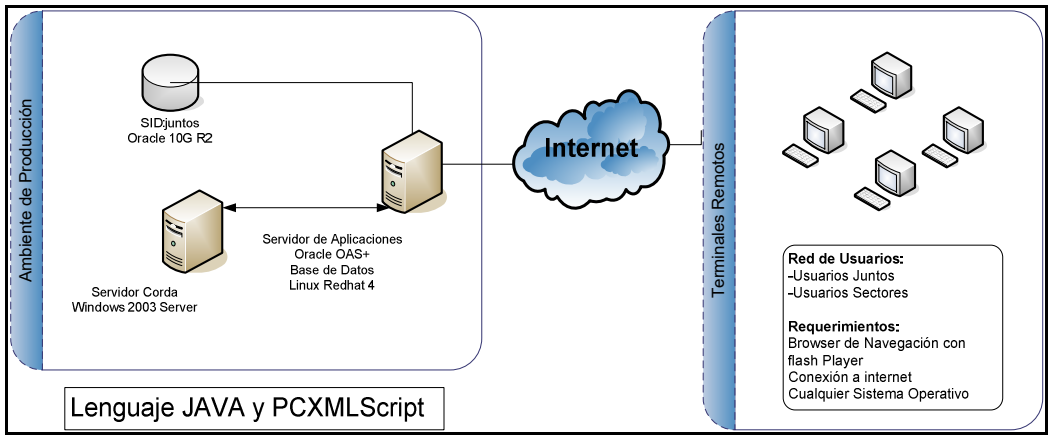


Figura 17. Arquitectura Portal Web

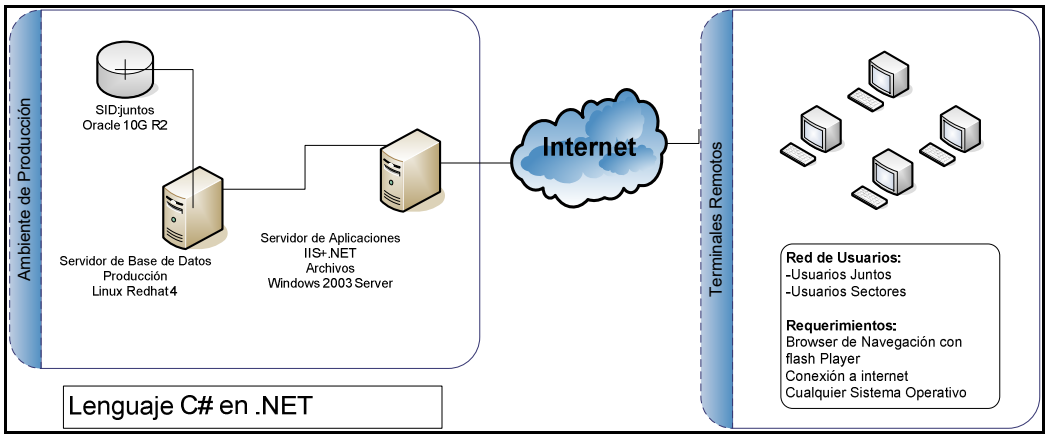


Figura 18. Arquitectura para escaneo

4.4.2 Entregables

Dentro del proceso de desarrollo de software dentro de JUNTOS se producen los siguientes artefactos:

1. Modelo de negocio.
2. Requerimientos.
3. Análisis y diseño.
4. Arquitectura de software.
5. Implementación.
6. Despliegue.
7. Estándares de programación.
8. Manual de usuario.
9. Plan de pruebas.
10. Informe de pruebas.

4.4.3 Procedimiento pase a producción

Participantes según rol:

Rol	Descripción
Equipo del Proyecto GTI	Equipo de la GTI, encargado del seguimiento del proyecto. Aún si el proyecto es realizado por terceros, es necesario que exista un responsable de la GTI,

	encargado de supervisar el desarrollo del proyecto.
Equipo Desarrollador	Equipo de desarrollo del sistema, podría pertenecer a la GTI o ser un equipo contratado para tal fin por la GTI, o por la Gerencia usuaria.
Equipo CCS	Equipo de Control de Calidad de Software de la GTI
Unidad Orgánica Usuaria	Unidad Orgánica que ha requerido el desarrollo del sistema
Usuario Líder	Persona responsable del proyecto, por parte de la Unidad Orgánica Usuaria.
Equipo de Soporte Técnico	Personal del área de Operaciones de la GTI, tal como los DBA, administradores de red, etc.

Tabla 2. Participantes para el pase a producción

Este Procedimiento describe las actividades necesarias para la realización del pase a producción de un sistema informático nuevo. Para ello, es necesario que, después de haber realizado las pruebas de implantación y de aceptación del sistema, se disponga del entorno de producción perfectamente instalado en cuanto a hardware y software de base, componentes del nuevo sistema y procedimientos manuales y automáticos.

En función del entorno en el que se haya llevado a cabo el proceso de pruebas del sistema, habrá que instalar los componentes del sistema, en forma total o parcial.

También, se tendrá en cuenta la necesidad de migrar todos los datos o una parte de ellos. Una vez que el sistema ya está en producción, se les notificará a los responsables del procedimiento.

Pase a producción para nuevos desarrollos:

Para realizar el "pase a producción" de un sistema, se debe presentar los documentos que sustenten la implantación y aprobación del mismo, que fueron realizados durante el proceso de pruebas. Para ello, se debe considerar los siguientes documentos, dichos documentos, en su mayoría están descritos en el listado de Artefactos que conforman la Metodología de Desarrollo de Software utilizado en la GTI:

1. Modelo de negocio.- Documento que describe el Negocio y la necesidad que cubrirá el sistema o el cambio a un sistema, desde el punto de vista de tecnologías de información.

2. Requerimientos.- Documento que describe la necesidad de la Unidad Orgánica Usuaria.
3. Arquitectura de software
4. Análisis y Diseño
5. Implementación
6. Plan de pruebas
7. Acta de aceptación de la funcionalidad del sistema.- Acta elaborada luego de la ejecución de las pruebas usuarias y firmadas por éste (Usuario(s) Líder).
8. Informe de pruebas
    a. Informe de pruebas funcionales y técnicas: A cargo de CCS.
    b. Informe de pruebas de carga y estrés (Opcional): La GTI, deberá decidir si se realizan o no dichas pruebas y que equipo deberá realizarlas, tomando en consideración lo siguiente:
        i. De acuerdo al tipo de sistema y sus características, verificar si se cuenta con las herramientas de software necesarias, para realizar dichas pruebas.
        ii. Si el Equipo Desarrollador, es externo a Juntos o si es un desarrollo interno.
        iii. Las pruebas por defecto, se deberán realizar por el Equipo Desarrollador; sin embargo, de acuerdo a las características del sistema y coyuntura, estas podrían ser realizadas por el Equipo CCS y/o por el Equipo del Proyecto GTI.

De realizarse dichas pruebas, el entregable será: "Informe de pruebas de carga y estrés", documento con las pruebas del comportamiento en cuanto a rendimiento y procesamiento del servidor en el cual está instalada la aplicación y la base de datos.

Pase a producción para mantenimiento de sistemas:

Se deberán considerar únicamente los siguientes documentos, cabe señalar, que de considerarse necesario, la GTI puede exigir mayor documentación.

1. Requerimientos (Solicitud de requerimiento aprobada).

2. Análisis y Diseño.
3. Implementación.
4. Plan de pruebas.
5. Acta de aceptación de la funcionalidad del sistema (área usuaria).
6. Informe de pruebas funcionales y técnicas.
7. Manual de Operación o de Usuario actualizado
8. Plantilla de solicitud de pase a producción.
9. Cargo que certifica la entrega de las fuentes del sistema.

4.5 Evaluación comparativa entre las herramientas tecnológicas

4.5.1 Comparativa ente OOAD y SOAD

De Componentes a Servicios	
Fuertemente acoplado: requiere una librería de tipos en el cliente para poder invocarlo remotamente.	**Débilmente acoplado**: intercambio de mensajes y configuraciones por políticas. Se usan clases proxies.
Comunicación remota orientada a **funciones**.	Comunicación remota orientada a **mensajes**.
Orientación a objetos en todos los niveles.	**Orientación a mensajes** externamente. **Orientación a objetos** internamente.
Implementación de servidor conocida.	Abstracción (servicios como cajas negras).
Cliente servidor.	Posibilidad **peer-to-peer** además de **cliente servidor**.
Expandible.	**Combinable** además de expandible.
Recomendable **sin estados**.	**Independiente** del contexto del consumidor.
Rápido.	Más **pesado** si se utilizan estándares incompatibles (SOAP, XML, HTTP).
Diseño de interfaces con **granularidad pequeña-mediana**.	Diseño de interfaces con **granularidad mediana-gruesa**. Hay que tener en cuenta el costo de latencia de llamadas

	remotas, por lo que tenemos que minimizar el número de llamadas a los servicios.
Construido **para que dure**.	Construido **para que cambie**.
Ciclos de desarrollo **prolongados**.	Construcción y despliegue **incremental**.

Tabla 3. Comparación entre OOAD Vs. SOAD

4.5.2 Comparativa entre OpenESB y ServiceMix

ServiceMix es un Enterprise Service Bus (ESB), basado en la especificación JBI puede funcionar como un servicio independiente (standalone) o como un servicio dentro de otro ESB. Podemos usar ServiceMix en Java SE o en Java EE Application Server.

- ServiceMix incluye un completo contenedor JBI que soporta todos los aspectos de la especificación JBI, incluidos:
- Servicio de mensajes normalizados y enrutado (NMR – Normalized Message Service and Router).
- JBI Management Mbeans.
- Ant task para la administración e instalación de componentes.
- Soporte completo para despliegue de unidades JBI con despliegue en caliente de los componentes JBI.

OpenESB: OpenESB es una implementación de un ESB basado en la especificación JBI, iniciada por Sun Microsystems. Permite integrar fácilmente aplicaciones empresariales y web services como aplicaciones compuestas débilmente acopladas. Esto permite componer y recomponer de manera fluida y rápida aplicaciones compuestas, con todas las ventajas de una verdadera Arquitectura Orientada a Servicios. Incluye una gran variedad de componentes JBI, como SOAP-over-HTTP-binding, y un motor de servicio WS-BPEL 2.0. Además incluye su propio motor BPEL (BPEL SE). Debido a la arquitectura débilmente acoplada de Open ESB es posible cambiar este motor por cualquier otro que cumpla la especificación JSR 208, como por ejemplo el de ServiceMix.

La integración con el entorno de desarrollo NetBeans permite el despliegue de aplicaciones de una manera rápida y eficiente, con una serie de facilidades como ayuda en el desarrollo, control de errores y pruebas.

Pensando en la escalabilidad y desarrollo del EVT, sería deseable la utilización de OpenESB ya que presenta ostensibles mejoras sobre su competidor en lo que a orquestación de servicios se refiere. Su punto fuerte reside en la novedosa interfaz gráfica proporcionada a través de NetBeans, que hace que la tarea de desarrollar servicios web, guiones BPEL y paquetes JBI sea extremadamente simple en comparación con el bus desarrollado por Apache.

Ventajas de Open ESB frente a Servicemix:

Integración con servidor de aplicaciones Glassfish de Sun.

Open ESB emplea gran empeño en el desarrollo de herramientas que faciliten su uso. Para ello puede ser integrado en un gran entorno de desarrollo usando la interfaz gráfica de Netbeans que permite la creación de aplicaciones SOA capaces de ser desplegadas con facilidad en el bus de integración. En el caso concreto de orquestación de servicios, cuenta además con un gran editor de guiones bpel, asi como editores de documentos xsd y wsdl, capaces de facilitar en gran manera el desarrollo de este tipo de apliaciones.

Integración con diversidad componentes. En su última versión cuenta con cuatro Services Engines (BPEL, XSLT, IEP, Java EE) y ocho componentes Binding. Además el equipo de desarrollo de OpenESB se encuentra desarrollando un gran número de componentes adicionales.

Presenta el motor JavaEE (inexistente en Servicemix), de gran importancia ya que permite la integración entre los distintos componentes JBI y webservices existentes en Glasfish utilizando el motor BPEL.

Gran cantidad de documentación sobre su uso, con ejemplos prácticos que hacen que la curva de aprendizaje sea mucho más suave.

Inconveniente OpenESB frente a ServiceMix

El gran inconveniente de OpenESB frente a Servicemix es la incapacidad de ser ejecutado como un contenedor independiente (standalone), existiendo hasta ahora la única posibilidad de ser integrado en Glassfish, si bien en este aspecto se está trabajando en ambos sentidos, siendo posible en un futuro ser ejecutado como un contenedor independiente o ser integrado en otros servidores de aplicaciones como Jboss.

4.6 Selección de la herramienta de tecnológica

Se considerando la evaluación entre las herramientas para el desarrollo del Entreprise Service Bus: OpenESB y ServiceMix.

4.6.1 Factores de decisión

Los principales factores para tomar la decisión de qué herramienta tecnológica utilizar para el desarrollo del ESB, según el ámbito del problema a resolver, es:

- Tiempo de desarrollo del software integrador, para poder cumplir con plazos establecidos y responder con facilidad ante el cambio en el negocio.
- El ESB debe estar integrado a algún Entorno de Desarrollo Integrado (IDE), para no invertir tiempo en configurar el IDE con los plugins necesarios para un correcto funcionamiento.
- El IDE debe tener una interfaz grafica para modelar los procesos de negocio en BPEL o BPM, de esta forma la orquestación es más sencilla y se puede tener una mejor visión de todos los elementos involucrados.
- El IDE debe tener la capacidad de realizar pruebas, para poder comprobar si la orquestación está arrojando valores esperados.
- Tener gran cantidad de documentación sobre el uso y aplicaciones prácticas.

Tomando en cuenta los factores anteriores y la comparativa entre OpenESB y ServiceMix desarrollado en el punto 3.5.2, se ve conveniente el uso de OpenESB como herramienta para la construcción del ESB.

4.7 Contribución teórica o adaptación de la herramienta tecnológica

Tomando como premisa el problema que se intenta resolver, integrar aplicaciones mediante un bus empresarial, y no aplicar por completo una metodología SOA se ve conveniente adaptar el "Análisis y Diseño Orientado a Servicios", debido que cubre mejor las necesidades del problema en contraparte al "Análisis y Diseño Orientado a Objetos" que solo se crearán los servicios web en los sistemas orientados a objetos.

La adaptación para el Enterprise Service Bus OpenESB es la siguiente:

La metodología empleada para el desarrollo de Grupos de Trabajo utilizando como Bus de Integración Open ESB fue la siguiente:

1. Revisar la documentación de los sistemas existentes para ver si cumplen criterios mínimos de Ingeniería de Software para su construcción y realizar un estimado de tiempo para las modificaciones de las clases necesarias para convertir los métodos en web service.
2. Modificar, en caso sea necesario, las aplicaciones para que respondan entidades estándares y sencillas (POJO), acá se debe trabajar principalmente con las clases "Manager" o en las clases "Interface" donde se encuentra la secuencia de acciones para atender una funcionalidad especifica del software, no manipular las clases donde se encuentren la interfaz de usuario. Para un mejor trabajo es recomendable que la aplicación a integrar cuenta con patrones de desarrollo, por ejemplo MVC, DAO, etc.
3. Generar los servicios web de las funcionalidades necesarias contenidas en las clases detalladas en el paso anterior, para este fin una gran cantidad de IDEs proporcionan facilidades para la creación de web service desde los métodos contenidos en las clases y su posterior despliegue en el servidor de aplicaciones.
4. Creación de la secuencia (orquestación) de servicios, mediante BPEL, que utilizarán los servicios web creados y desplegados anteriormente. Para lograr este fin la mayoría de IDEs proporcionan una interfaz gráfica mediante diagramas (BPEL) similar a la notación BPMN y detrás de este entorno gráfico se genera código XML que será entendido por el ESB.
5. Generar el Service Assembly utilizando el estándar JBI. Este se realizará de forma automática en un sencillo paso utilizando un asistente del IDE. El propio IDE generará todos los descriptores correspondientes necesarios para su posterior despliegue en el Bus de Integración.
6. Desplegar en el ESB, al igual que en pasos anteriores a través del IDE. Una vez desplegados, estarán disponibles los Web Service Description Language (WSDL), tanto de la orquestación BPEL como de los Web Services, accesibles mediante en la Web a través de una URL.
7. Finalmente, para consumir los servicios web desde una aplicación cliente se utiliza el Soporte de Java API para XML Web Services (JAX-WS). El cliente final consistirá en una serie de clases cuyos métodos coincidirán con las operaciones

del web service u orquestación BPEL a invocar. Este paso también se realizará de forma automática utilizando un IDE para desarrollo Java o .NET.

Capítulo 5: Aporte práctico

Se presenta los documentos de diseño de la solución planteada en el capitulo anterior.

5.1 Identificación de los principales macro procesos operativos

El Programa JUNTOS trabaja bajo el siguiente ciclo de procesos operativos:

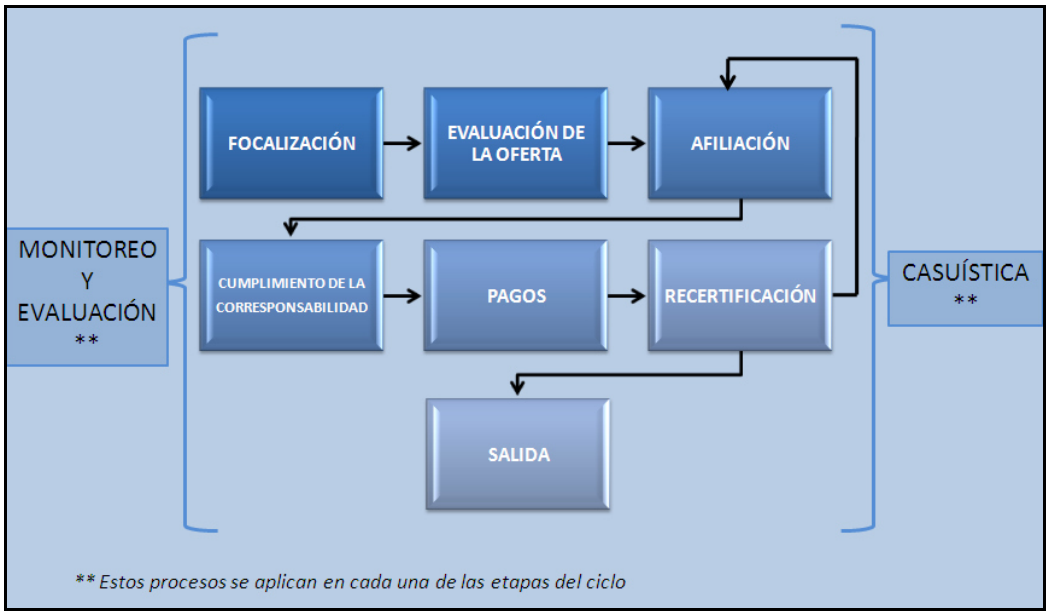


Figura 19. Macro Procesos Operativos de JUNTOS

Siendo los principales procesos operativos a destacar:

- Focalización.
- Afiliación de Hogares.
- Verificación del cumplimiento de corresponsabilidades.
- Transferencia Monetaria.
- Servicio al Usuario.

5.1.1 Focalización

Proceso de identificación de las regiones de intervención y en éstas, de los hogares elegibles para el Programa JUNTOS. La focalización se realiza en dos

etapas de carácter secuencial (i) Geográfica, determina si el Programa interviene ó no en un área; e, (ii) Individual, en el área geográfica determinada se eligen los hogares.

i. Focalización Geográfica

Tiene en dos partes, la primera se selecciona a los distritos candidatos de acuerdo al índice ponderado de pobreza. Luego, dentro de los distritos seleccionados se identifican los centros poblados que están dentro de las áreas de influencia de los servicios de salud o educación. Asimismo, permite identificar centros poblados donde se impulse la priorización de nueva oferta de servicios en el marco de un enfoque de derechos.

ii. Focalización Individual

Se realiza por única vez a cada hogar, al inicio de la intervención del Programa. Se identifican los hogares en extrema pobreza utilizando la Ficha Socioeconómica Única FSU del SISFOH.

Mientras que SISFOH y el Gobierno Municipal se hagan cargo de la focalización, el Programa JUNTOS asume este proceso.

El proceso de focalización se desarrolla bajo dos modalidades 1) Modalidad Masiva ó por oferta; y, 2) Modalidad individual continua ó por demanda.

1. Modalidad de Focalización Masiva o por Oferta

Consiste en la realización del barrido censal, es decir se aplica la FSU a la totalidad de los hogares de una determinada localidad.

2. Focalización Individual –Modalidad Continua o por Demanda

Aquellos hogares que por circunstancias específicas no pudieron ser entrevistados durante el barrido censal, pueden presentarse a las Oficinas de la Ventanilla Social Local para su levantamiento de información respectiva y posibilitar su adscripción al Programa JUNTOS ó de cualquier otro Programa Social.

5.1.2 Afiliación de hogares

Proceso de identificación de los Miembros objetivo de los ***hogares elegibles*** e inscribirlos en el Programa JUNTOS. El procedimiento de Afiliación de hogares se inicia cuando el Equipo de Focalización y Recertificación del Programa JUNTOS entrega los listados de **hogares pobres extremos** por distrito al Equipo de Afiliación.

El proceso de afiliación de hogares tiene las siguientes actividades:

- Evaluación de capacidad de oferta de servicios de salud y educación.
- Selección de hogares elegibles.
- Cálculo de requerimientos presupuestales.
- Planificación logística.
- Asamblea de validación comunal.
- Difusión del proceso de afiliación.
- Evento de Afiliación.
- Registro, verificación, reporte y confirmación la adscripción de los hogares.

5.1.3 Verificación del cumplimiento de corresponsabilidades

Los procesos de control del cumplimiento de las corresponsabilidades del Programa JUNTOS se basa en el seguimiento y registro histórico de:

- las atenciones en salud que reciben las gestantes y los menores de 5 años y que realiza el personal de salud; y,
- los reportes que hacen los docentes de las instituciones educativas sobre la promoción, matrícula y asistencia a clases de los niños/niñas entre 5 y 14 años.

En el desarrollo de esta actividad intervienen de forma activa MINSA y MINEDU ya que su personal es el encargado de reportar al Programa JUNTOS el cumplimiento de las corresponsabilidades de los miembros objetivo.

5.1.4 Transferencia monetaria

Proceso de entrega de incentivos monetarios a los hogares adscritos al Programa JUNTOS que se inicia al completar el proceso de afiliación y continúa en períodos bimestrales. La entrega de estas transferencias está condicionada al cumplimiento de corresponsabilidades. Eventualmente puede incluir la restitución de transferencias, previo reclamo reconocido.

5.1.5 Servicio al usuario

Los requerimientos, demandas y reclamos de los usuarios del Programa JUNTOS son atendidos a través de un proceso que se inicia con una solicitud o pedido de los hogares adscritos; también canaliza las limitaciones de la oferta de servicios, la afiliación, focalización, verificación de condicionalidades, transferencias monetarias, acreditación, recertificación entre otros relacionados con los procedimientos del Programa JUNTOS.

[JUN 10]

5.2 Descripción del proceso de transferencia monetaria

Se selecciona el proceso de transferencia monetaria para detallar hasta un nivel de detalle mínimo como se realiza este proceso. Con el fin de luego plasmarlo en el ESB.

La secuencia de pasos se puede apreciar en el diagrama de actividades "Transferencia Monetaria" el detalle de este caso de uso de negocio se puede apreciar en el Anexo I y las reglas del negocio (Business rules) que sigue la transferencia monetaria se detalla en el Anexo II.

De todo el conjunto de actividades de "Transferencia Monetaria" desde el punto de vista del negocio se desarrollará la actividad **"Generar Padrón de Beneficiarios"**, esta tarea es una tarea a automatizar (ver Anexo I).

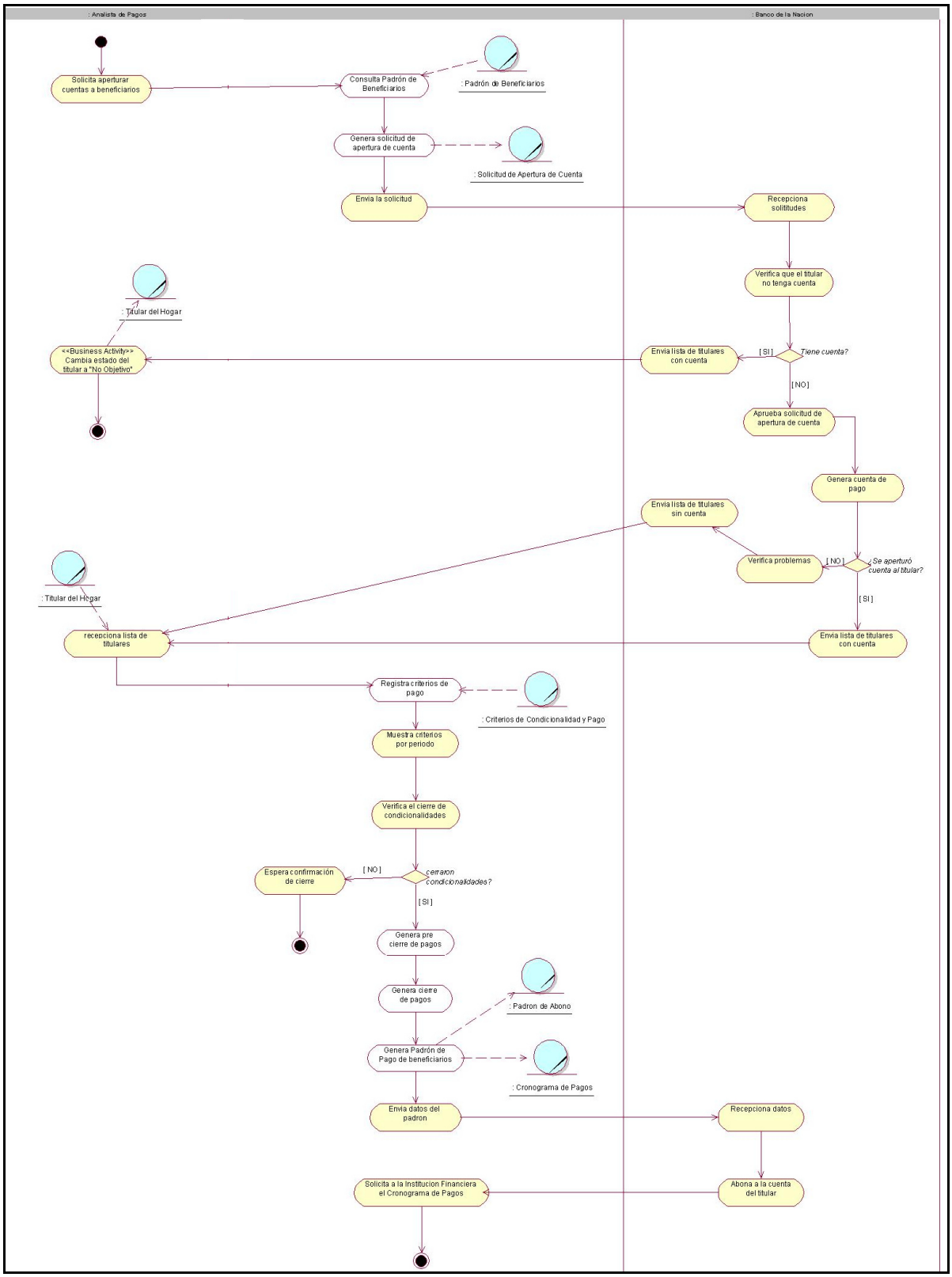


Figura 20. Diagrama de Actividades del Procesos de Negocio: Transferencia Monetaria

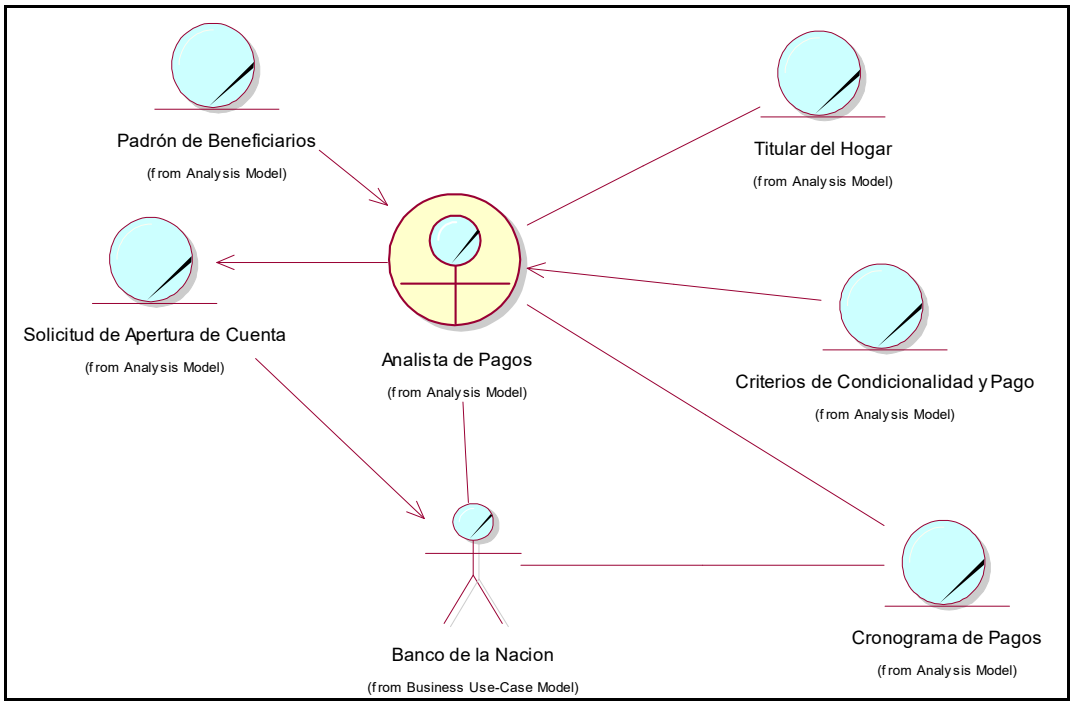


Figura 21. Diagrama de Clases de Negocio: Transferencia Monetaria

Para la Actividad Generar Padrón de Beneficiario (Actividad a desarrollar en el presente trabajo) tienen las siguientes reglas y/o pasos:

1. El titular del hogar (y de la cuenta bancaria) debe tener DNI validado y habilitado: esto es necesario para realizar cualquier trámite.
2. Todos los hogares del Padrón Activo deben tener al menos un miembro Objetivo: donde miembro objetivo es un niño o niña entre 0-14 años de edad y/o exista una gestante.
3. Los hogares que van al Padrón Pasivo son aquellos que tienen su titular afiliado, con cuenta bancaria y que tienen suspensión temporal: la suspensión temporal se realiza en el macroproceso Verificación de Cumplimiento de Corresponsabilidades.
4. Registrar al hogar en el padrón activo o pasivo.

5.3 Estándares a utilizar en la aplicación a desarrollar

Los estándares a utilizar en la aplicación son:

- Casos de Uso de Negocio

- SOAD
- WSDL
- XML
- SOAP
- JBI
- NMR
- BPEL

Todos estos términos se detallan en el capítulo 2 (marco teórico).

La descripción de los procesos de negocio se realiza mediante los casos de uso de negocio.

El análisis y diseño orientado a servicio se aplica para establecer los medios de comunicación entre los diversos servicios de la aplicación compuesta.

Java Business Integration (JBI), donde los componentes aparecen en forma de plug-in´s conectados a contenedores JBI y actuando como proveedores de servicio o consumidores de servicio. Service engines maneja la lógica de negocio y Binding components que aisla al entorno JBI del heterogéneo exterior normalizando los mensajes entrantes y desnormalizando los mensajes salientes, utilizando Normalized Message Router (NMR).

Para introducir la funcionalidad al ESB, se realiza mediante web service que esta almacenado en los servidores que proporcionan los servicios, el ESB necesita conocer el WSDL para poder utilizar el web service.

La orquestación de los procesos se realiza mediante BPEL (Lenguaje de Ejecución de Procesos de Negocio) similar a BPMN, la entrada y salida del modulo BPEL es mediante SOAP (para el caso de estudio).

Para el ingreso de datos al proceso de negocio (procesos de transferencia monetaria para el caso de estudio) se crea el WSDL de tipo SOA que tiene especificado el formato de entrada a la funcionalidad que da soporte al proceso de negocio y de salida de mensajes que alimentaran a otros sistemas.

5.4 Creación de los Web Service

Para la creación de estos servicios se realizan dentro de la capa de la lógica del negocio en la arquitectura del software que se mostró en el capitulo anterior.

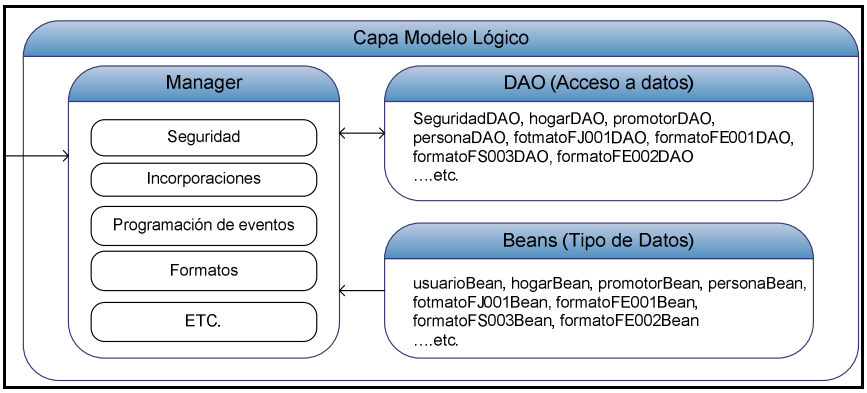


Figura 22. Capa Modelo Lógico

Para validar el número de DNI es necesario consumir el servicio web proporcionado por la RENIEC:

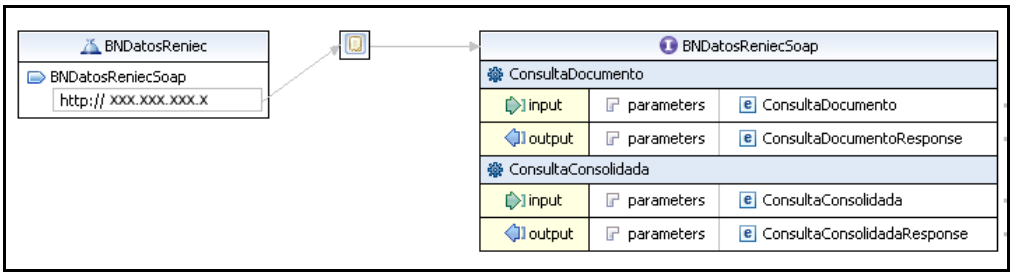


Figura 23. Web Service RENIEC

La validación del código SIS y si una madre es gestante o no se realiza mediante el Web Service del SIS.

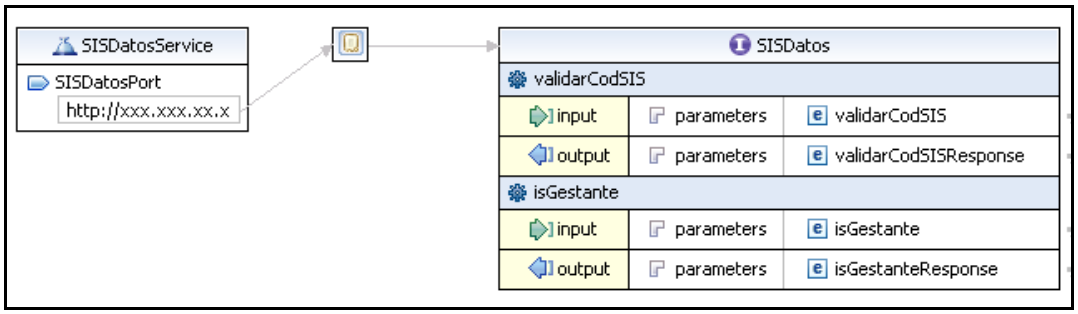


Figura 24. Web Service SIS

Web Services generados desde los sistemas de JUNTOS:

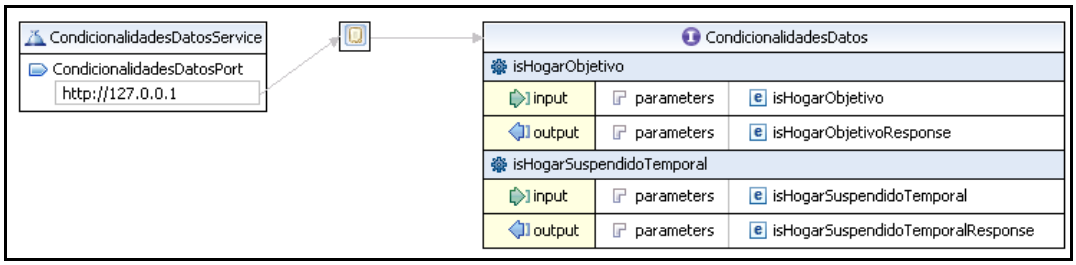


Figura 25. Web Service Condicionalidades

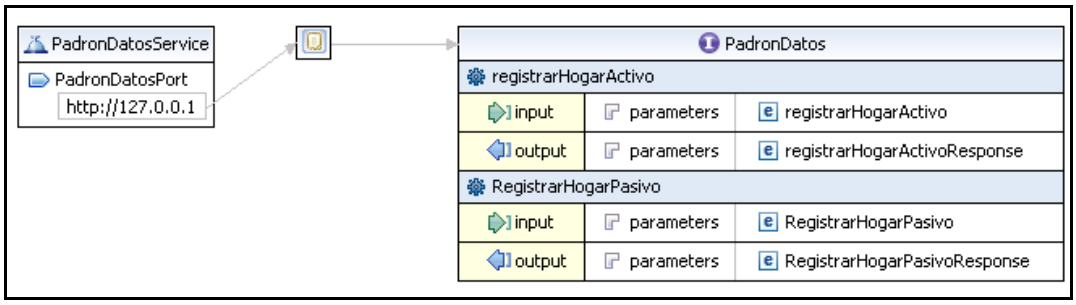


Figura 26. Web Service Padrón de Beneficiarios

5.5 Orquestación de procesos

El primer paso antes de orquestar los procesos, es importar los web service al proyecto BPEL, posteriormente se debe crear la secuencia de pasos a seguir mediante BPEL.

Para el ingreso de datos al proceso de transferencia monetaria se crea el WSDL que tiene especificado el formato de entrada y de salida de mensajes que alimentaran a otros sistemas.

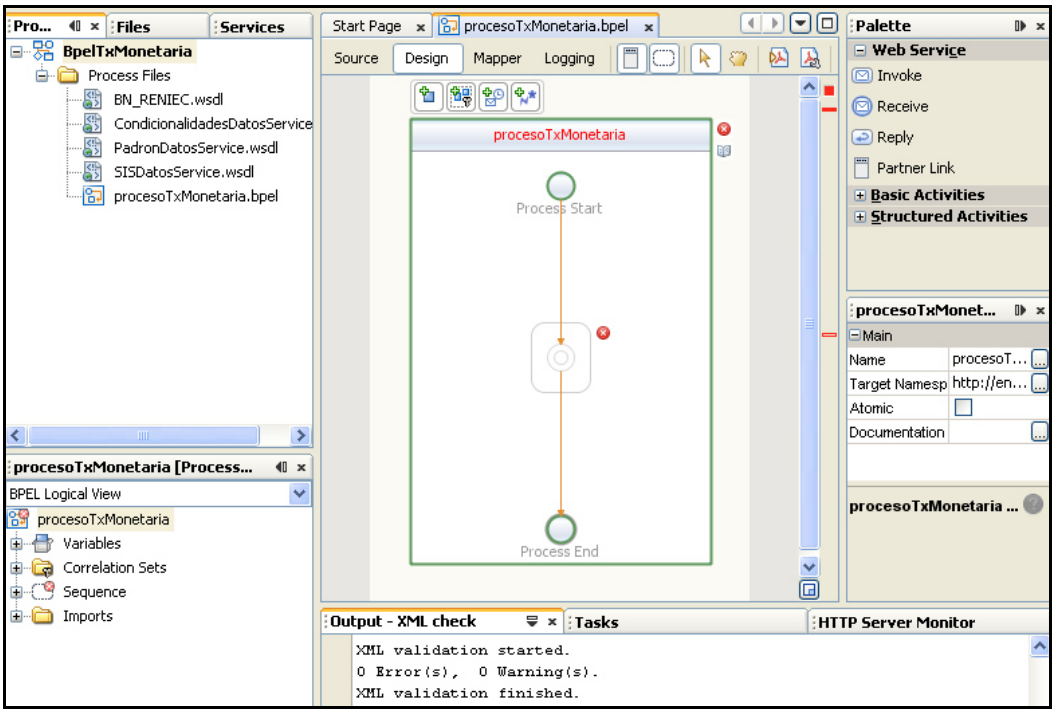


Figura 27. Creación BPEL

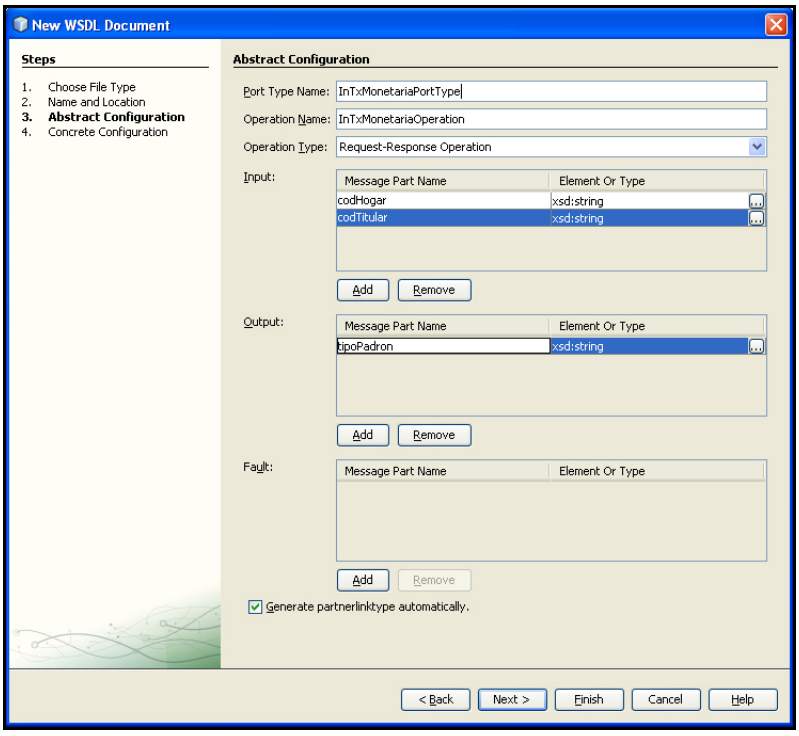
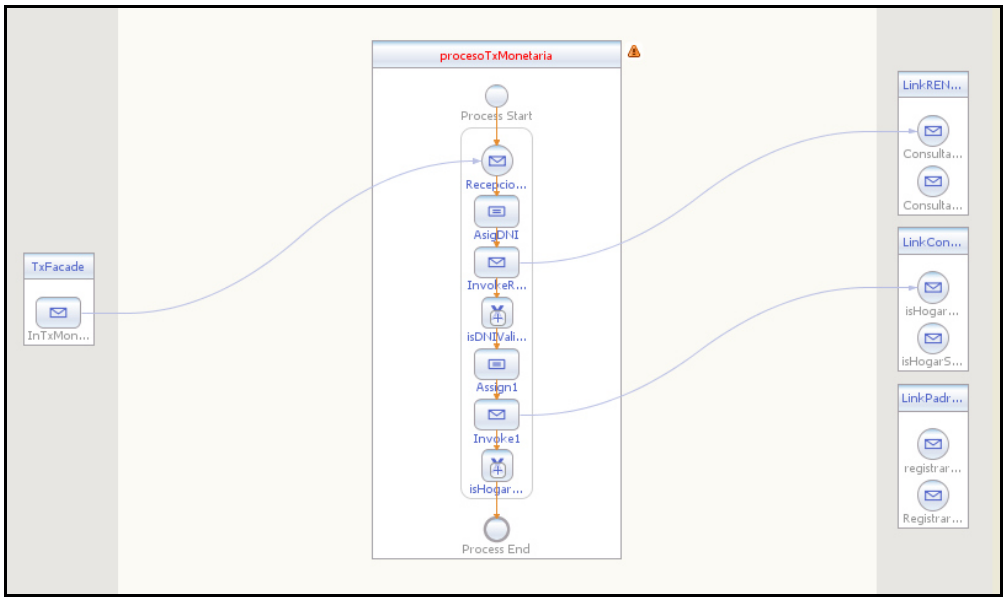
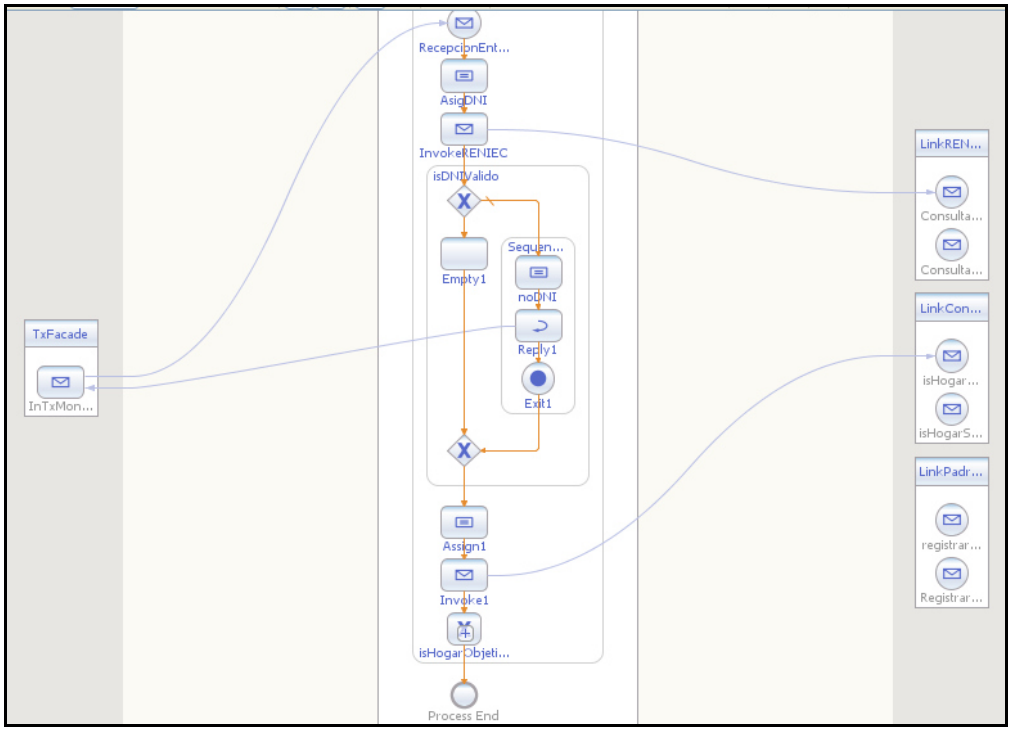


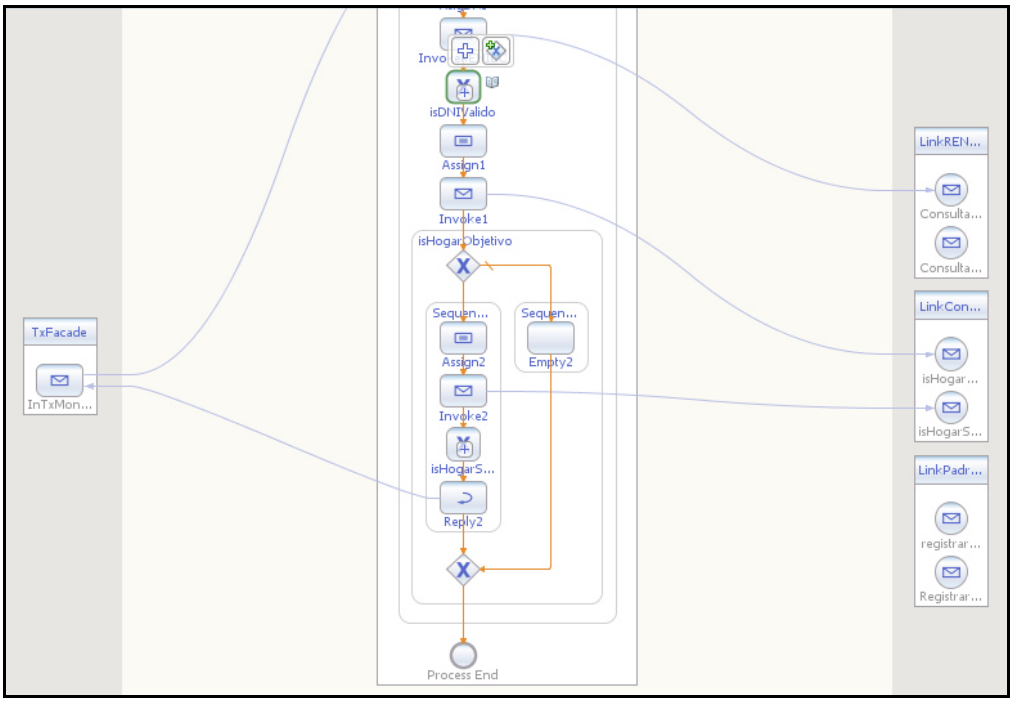
Figura 28. In/Out BPEL

La entrada y salida del BPEL llamada "InTxMonetariaBinding" es del tipo SOAP, también pudiendo ser File, HTTP, JMS.

Distintas actividades para el proceso de Transferencia Monetaria.







Finalmente se construye el componente (.jar) que será utilizado en la composición de aplicaciones.

5.6 Composición de aplicaciones

Para la composición de aplicaciones se debe agregar todos los módulos JBI que se tengan además estos módulos se pueden comunicar en este punto.

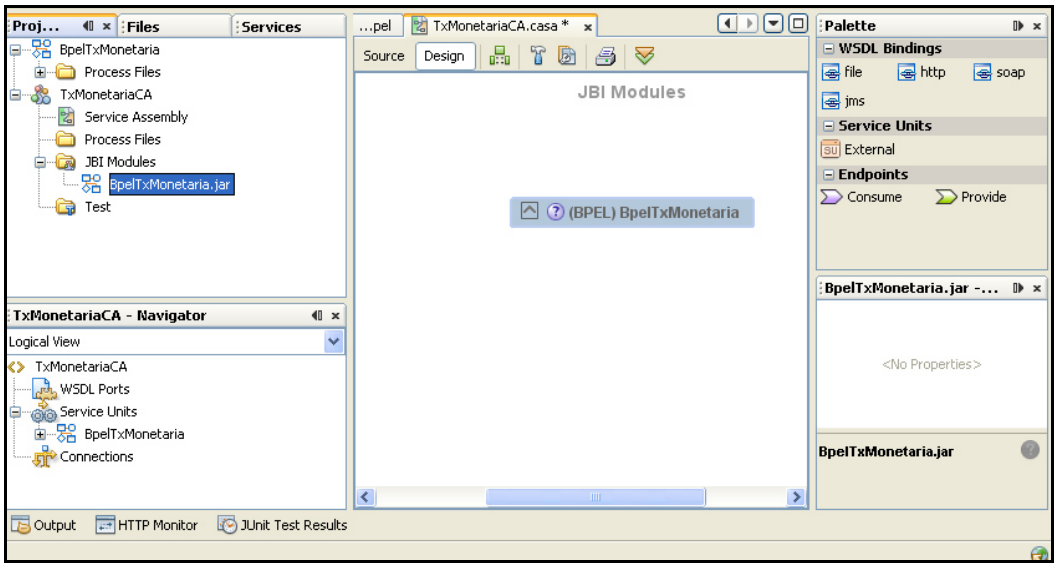
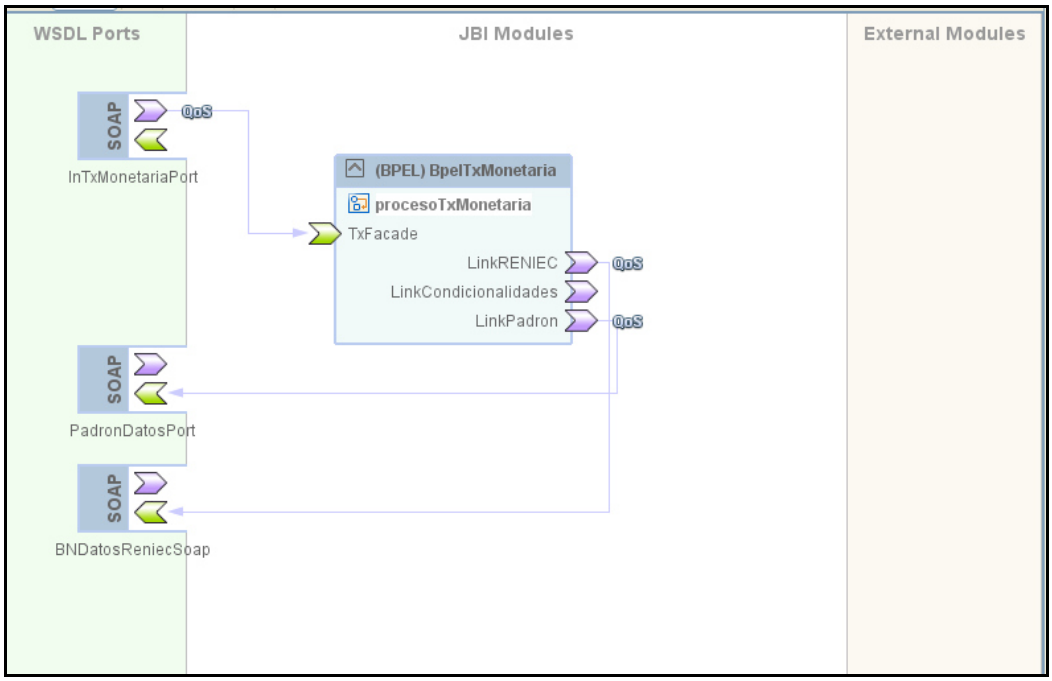


Figura 29. Adicionar Modulo JBI

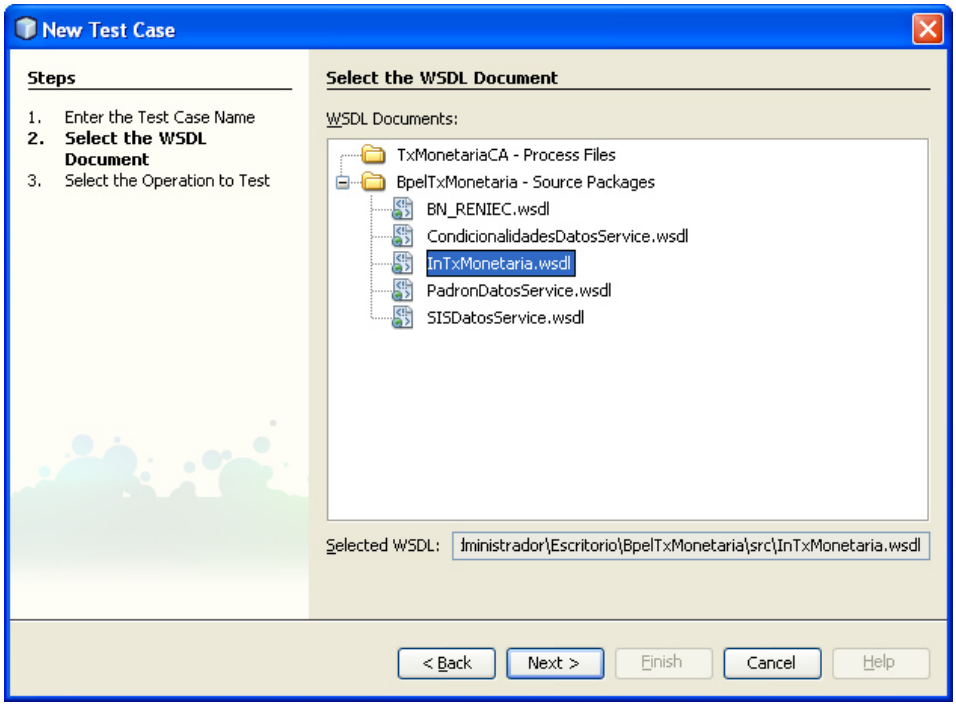
Teniendo el modulo JBI para la composición de la aplicación, se debe crear el enlace WSDL que puede ser del tipo File, HTTP, SOAP, JMS, en este caso se utiliza el de tipo SOAP.

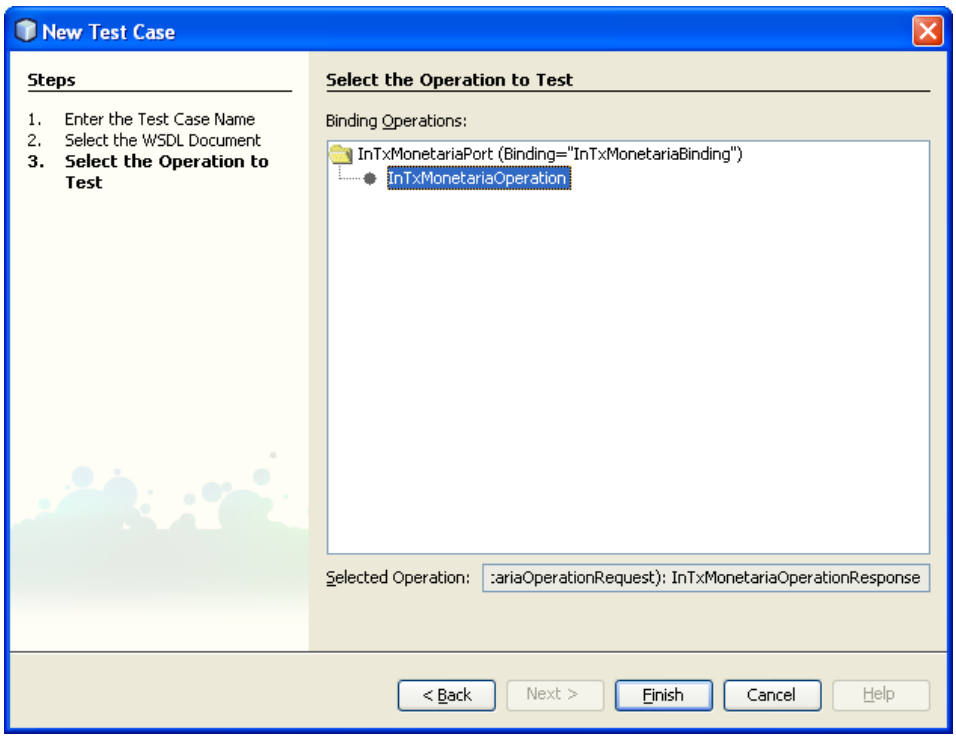


Hasta este punto se tiene los puntos de acceso a la solución integradora, solo faltaría probar su correcto funcionamiento.

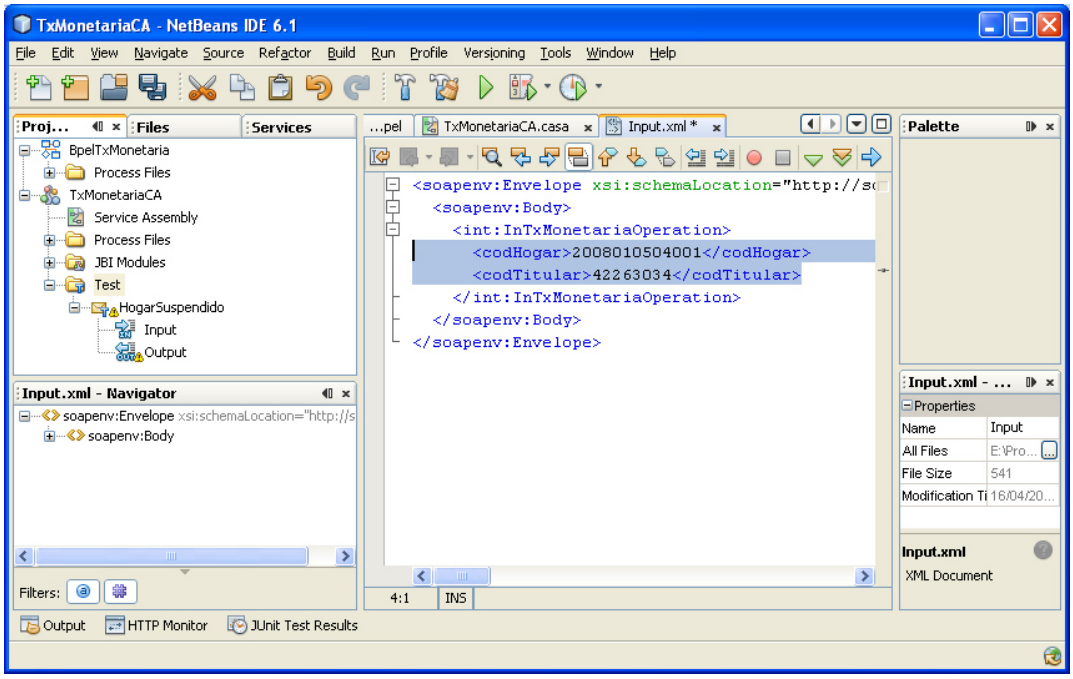
Creando casos de pruebas se puede probar si la solución integradora da como resultado los datos esperados.

Como primer paso para las pruebas se debe crear el caso de prueba, luego seleccionar el documento WSDL (el esquema proporciona los datos a ingresar) que sirve de entrada al ESB y finalmente la operación del WSDL a utilizar.

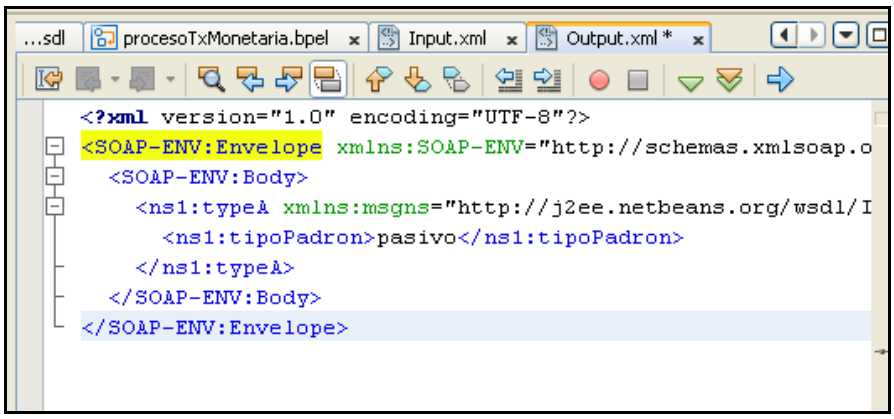

New Test Case
Steps
1. Enter the Test Case Name
2. Select the WSDL Document
3. Select the Operation to Test
Select the WSDL Document
WSDL Documents:
TxMonetariaCA - Process Files
BpelTxMonetaria - Source Packages
BN_RENIEC.wsdl
CondicionalidadesDatosService.wsdl
InTxMonetaria.wsdl
PadronDatosService.wsdl
SISDatosService.wsdl
Selected WSDL:
Iministrador\Escritorio\BpelTxMonetaria\src\InTxMonetaria.wsdl
< Back
Next >
Finish
Cancel
Help


New Test Case
Steps
1. Enter the Test Case Name
2. Select the WSDL Document
3. Select the Operation to Test
Select the Operation to Test
Binding Operations:
InTxMonetariaPort (Binding="InTxMonetariaBinding")
InTxMonetariaOperation
Selected Operation:
:ariaOperationRequest): InTxMonetariaOperationResponse
< Back
Next >
Finish
Cancel
Help

Solo falta ingresar los datos de prueba



El resultad seria el esperado (padrón pasivo), con lo cual, se termina el caso de prueba satisfactoriamente.



Capítulo 6: Experimentos con estudio de casos

En este capítulo se muestran los resultados obtenidos al aplicar el ESB para la integración de aplicaciones en el caso de estudio (JUNTOS) y se realiza la comparación con el modelo antes existente.

6.1 Reducción de tiempos de desarrollo

Como se observó en el desarrollo de la solución para el caso de estudio, se reutiliza funcionalidad almacenada en sistemas heterogéneos y repartidos en distintas ubicaciones geográficas. De esta forma el tiempo de desarrollo disminuye pero la planificación y el diseño de la arquitectura se deben realizar con anticipación y de forma detallada para que no existan distintos servicios que persigan el mismo objetivo de negocio.

Para el caso de estudio el tiempo de desarrollo para la integración llevo alrededor de medio día, con cierta pericia en el manejo de las herramientas y conceptos previos. El mismo problema resuelto en el paradigma orientado a objetos demoraría un día configurando el servicio de nombres e interfaces de forma adecuada y no directamente en el código.

6.2 Incremento de la calidad del software

La calidad del software, entendida como la satisfacción de las necesidades de las organizaciones en el plazo y presupuesto adecuado, tiene en el paradigma de integración de aplicaciones la misma importancia que en un entorno de software tradicional, contemplando la medición y el control de un amplio conjunto de parámetros ligados a la funcionalidad, escalabilidad y robustez de los aplicativos.

Por las razones antes expuestas se puede ir migrando el software de un paradigma íntegramente orientada a objetos a uno orientado a servicios, debido que ambas arquitecturas pueden convivir e interoperar.

6.3 Software adaptable a los cambios del negocio

Debido que toda la funcionalidad (puntual), con un grano grueso, esta almacenada en los servicios es posible orquestar todos estos servicios para que respondan a las necesidades de un proceso de negocio.

Los servicios pueden ser proporcionados por desarrollo en casa o tercerizado solo se necesita establecer las interfaces de datos (esquemas) y los estándares para la nomenclatura de los servicios, esto se define en el documento "Estándares de desarrollo" definido por el área de TI de la organización o por una consultora, estos servicios se pueden orquestar mediante BPEL en una consola centralizada para facilitar el monitoreo y futuras modificaciones del mismo.

El aspecto tratado se puede visualizar cuando se crea el flujo que sigue el orquestamiento de los servicios web donde se puede insertar iteraciones y flujos de control.

Capítulo 7: Conclusiones y trabajos futuros

La base de la arquitectura orientada a servicios es la autonomía de los servicios. Los servicios están desacoplados los unos de los otros, no hay dependencia tan cercana como en la programación orientada a objetos.

Debido a la gran diversidad de entornos donde podrían usarse nuestros servicios, se debe tender al uso de estándares que faciliten la integración. Así mismo no se debe nunca confiar en que los mensajes de entrada que estarnos recibiendo son válidos, debemos siempre realizar comprobaciones muy estrictas de la información que se recibe.

El uso de servicios puede transformar los procesos empresariales en servicios reutilizables y flexibles, lo que permite mejorar y optimizar los nuevos procesos.

El uso del ESB es el punto de entrada para distintas aplicaciones para conectar la infraestructura con eficiencia, integrando a todas las personas, procesos e información de la organización. Al tener conexiones flexibles entre los servicios y el todo el entorno, se puede tomar un procesos de negocio ya existente y ofrecerlo sin mucho esfuerzo a través de otro canal empresarial.

En el estudio de caso se pudo observar que el tiempo de desarrollo disminuye siempre y cuando exista diseño y planificación en la arquitectura a desarrollar, además que el equipo tenga pericia en el manejo de las herramientas y conceptos acerca de orientados a servicios. Se puede ir migrando el software de un paradigma orientado a objetos a uno orientado a servicios de forma incremental debido que ambas arquitecturas pueden convivir e interoperar.

Comprendiendo como integrar las diversas aplicaciones se podría dar el siguiente paso que es migrar a una arquitectura 100% SOA la que está dirigida a la gestión de procesos empresariales por medio del Business Process Management (BPM), de esta forma se está pasando del ESB a SOA unida a la gestión dirigida por procesos y posteriormente a una arquitectura de negocio (empresarial). Es decir la arquitectura orientada a servicios, desde la perspectiva SOA, permitirá unir personas, procesos, información, conectividad y reutilización.

REFERENCIAS BIBLIOGRÁFICAS

[TOR 05] Torres C., El Proyecto de Investigación Científica, Libros y Publicaciones, tercera edición, 2005, p. 251

[CHA 04] Chappel D., Enterprise Service Bus, O'Reilly Media Inc., primera edición, 2004

[MIC 08] Arquitectura SOA con tecnologías Microsoft, Krasis Consulting S.L., primera edición en español, 2008, pp. 1-38

[KLO 07] Khoshafian S., Service Oriented Enterprises, Auerbach Publications, 2007, pp. 77-92

[BOO 08] Booch Jacobson Rumbaugh, El Lenguaje Unificado de Modelado, segunda edición, 2008

[JUN 10] JUNTOS, Manual de Operaciones, 2010, p. 64

[PRE 05] Pressman R., Ingeniería de Software, Mc Graw Hill, sexta edición, 2005, pp. 275- 348, p.958

[ORA 07] Oracle, Build Java EE Applications I, segunda edición, v. 2, 2007

[ORA 06] Oracle, Build Java EE Applications II, primera edición, v. 1, 2006

Anexo I: Especificación CUN Transferencia Monetaria

1.- Proceso de Negocio	Gestionar Pagos
2.- Introducción	6.1 **Alcance** Este proceso compete al Analista de Pagos y al Banco de la Nación quien realiza el abono mediante diversos medios. 6.2 **Definiciones, acrónimos y abreviaturas** El conjunto de Definiciones, Acrónimos y Abreviaciones usados se encuentran en el documento Glosario. 6.3 **Referencias** Las referencias obtenidas para plantear este caso de uso fueron obtenidas por entrevistas a los mismos trabajadores involucrados y por propia participación en el proceso.
3. Objetivo	Este caso de uso del negocio tiene como propósito, realizar el abono del incentivo monetario. En este proceso se apertura la cuenta bancaria del titular del hogar beneficiario, se coordina con el banco con el banco para asignar la modalidad del abono y el punto (ubicación de la agencia o punto de encuentro para la transportadora) de pago.
4.- Actores	Analista de Pago, Banco de la Nación.
5.- Precondiciones	
El hogar se encuentra registrado como hogar beneficiario y cumple con tas las condiciones (condicionalidades) impuestas por el programa social, firmada por el titular del hogar en el acuerdo de compromiso.	
6.- Flujo de Eventos	

6.1 Analista de Pago solicita aperturar cuentas de ahorro a los beneficiarios.

6.2 Analista de Pago consulta el Padrón de Beneficiarios.

6.3 Analista de Pago genera las solicitudes de apertura de cuenta por titular.

6.4 Analista de Pago envía las solicitudes a la Institución Financiera.

6.5 El Banco de la Nación recepciona las solicitudes

6.6 El Banco de la Nación verifica si los titulares tienen cuenta.

6.7 Si no tienen cuenta, la solicitud es aprobada.

6.8 El Banco de la Nación genera la cuenta de pago para cada titular.

6.9 Si se aperturaron las cuentas de pago, el Banco de la Nación envía una lista de titulares con cuenta.

6.10 Analista de Pago recepciona la lista de titulares.

6.11 Analista de Pago registra criterios de pago en el sistema.

6.12 Analista de Pago muestra criterios por periodo.

6.13 Analista de Pago verifica que se haya realizado el cierre de condicionalidades.

6.14 Si se realizó el cierre de condicionalidades, el Analista de Pago genera precierre.

6.15 Analista de Pago genera cierre.

6.16 Analista de Pago genera Padrón de Pago de Beneficiarios.

6.17 Analista de Pago envía datos del padrón al Banco de la Nación.

6.18 El Banco de la Nación recepciona los datos del padrón.

6.19 El Banco de la Nación abona a la cuenta del titular.

6.20 Analista de Pago solicita al Banco de la Nación el Cronograma de Pagos.

7.- Poscondiciones

Todo titular posee una cuenta bancaria.

Incentivo monetario abonado en la cuenta bancaria del titular del hogar.

Cronograma de pagos generado.

8.- Flujo Alternativo o Excepciones

- Si los titulares tienen cuenta, el Banco de la Nación envía una lista de titulares que ya contaban con cuenta y el Analista de Pago cambia el estado del titular a "No Objetivo".
- Si no se apertura la cuenta para algún titular, el Banco de la Nación verifica los problemas existentes y envía la lista de los titulares sin cuenta.
- Si no se realizo el cierre de condicionalidades, espera la confirmación del cierre y termina el proceso.

9.- Tareas a automatizar	Según el análisis que se hizo, se sugiere automatizar las siguientes actividades (ver diagrama de actividades): o Consulta Padrón de Beneficiarios o Genera solicitud de apertura de cuenta o Registra criterios de pago o Genera pre cierre de pagos o Genera cierre de pagos o Genera Padrón de pago de beneficiarios
10.- Versión	1.0
11.- Tiempo de Ejecución	

Anexo II: Reglas de Negocio para la Transferencia Monetaria

Proceso de Negocio	Actividad	Descripción de la Regla de Negocio
Gestionar Abonos	Aperturar cuenta bancaria	Se genera una lista de titulares que están afiliados y que no tienen Cuenta Bancaria Activa.
Gestionar Abonos	Aperturar cuenta bancaria	Se envían en un formato considerando que tenga sus datos completos, nombre, DNI, fecha de nacimiento, domicilio.
Gestionar Abonos	Aperturar cuenta bancaria	El banco crea una nueva cuenta sólo para los titulares cuyos DNIS, no tenga otra cuenta en el banco.
Gestionar Abonos	Aperturar cuenta bancaria	A los titulares que no se les apertura cuenta se debe indicar el motivo de la no apertura.
Gestionar Abonos	Crear Periodo del Padrón	Se apertura el Periodo del Padrón donde se deben indicar si el Mes y Año de Padrón
Gestionar Abonos	Crear Periodo del Padrón	Se debe certificar el monto que va a ser abonado por Hogar y los demás parámetros que pueden ser cambiados por periodo del Padrón.
Gestionar Abonos	Crear Periodo del Padrón	Se debe indicar si el Pago es de manera Ordinaria o Extemporánea
Gestionar Abonos	Generar Padrón de Beneficiarios	Los Hogares que van Padrón Activo son aquellos que tienen su Titular afiliado, con cuenta bancaria y que no tienen suspensión Temporal.
Gestionar Abonos	Generar Padrón de Beneficiarios	Todos los hogares del Padrón Activo deben tener al menos un miembro Objetivo.
Gestionar Abonos	Generar Padrón de Beneficiarios	Los hogares que van al Padrón Pasivo son aquellos que tienen su titular afiliado, con cuenta bancaria y que tienen suspensión temporal
Gestionar Abonos	Generar Padrón	Se clasifican los Hogares que están en el Pasivo, en el siguiente orden: suspendidos por Condicionalidad,

	de Beneficiarios	suspendidos por otras suspensiones temporales, por no incorporado en salud, y finalmente por no incorporado en educación.
Gestionar Abonos	Generar Padrón de Beneficiarios	Se clasifican los Hogares que no entran al padrón, por el siguiente orden: Por no estar validados por RENIEC, por no tener firma de convenio, por problemas en la apertura de cuentas, por no solicitar apertura de cuentas.
Gestionar Abonos	Generar Padrón de Beneficiarios	Los hogares que tienen al menos un niño de 0 a 3 años o una gestante se considera que está en la meta presupuestal 1, el resto de hogares están en la meta presupuestal 2.
Gestionar Abonos	Cerrar Padrón	Se debe mostrar todas las inconsistencias que se presentan de información como Alerta antes del Cierre de Padrón. Si no se aprueban las inconsistencias no se permitirá cerrar el Padrón.
Gestionar Abonos	Cerrar Padrón	Los hogares que no están incorporados en el Sector Salud, se suspende por 1 mes por motivo No incorporado en Salud
Gestionar Abonos	Cerrar Padrón	Los hogares que tienen niños de 6 a 14 años y que no están incorporados en el Sector Educación , se suspende por 1 mes por motivo No incorporado en Educación
Gestionar Abonos	Generar Padrón de Beneficiarios Extemporáneo	Los Hogares del Padrón Extemporáneo Activo no deben estar en el Padrón Activo Ordinario
Gestionar Abonos	Generar Padrón de Beneficiarios Extemporáneo	Al pasar los hogares al Activo extemporáneo se deben actualizar los resultados del padrón Ordinario